CRANFIELD UNIVERSITY

MOHD FADZIL FAISAE AB RASHID

INTEGRATED MULTI-OBJECTIVE OPTIMISATION OF ASSEMBLY SEQUENCE PLANNING AND ASSEMBLY LINE BALANCING USING PARTICLE SWARM OPTIMISATION

SCHOOL OF APPLIED SCIENCES

PhD
Academic Year: 2012 - 2013

Supervisor: Professor Ashutosh Tiwari
September 2013

CRANFIELD UNIVERSITY

SCHOOL OF APPLIED SCIENCES

PhD

Academic Year 2012 - 2013

MOHD FADZIL FAISAE AB RASHID

Integrated Multi-Objective Optimisation of Assembly Sequence Planning and Assembly Line Balancing using Particle Swarm Optimisation

Supervisor: Professor Ashutosh Tiwari
September 2013

This thesis is submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

ABSTRACT

In assembly optimisation, Assembly Sequence Planning (ASP) and Assembly Line Balancing (ALB) optimisations currently performed in serial, present an opportunity for integration, allowing benefits such as larger search space leading to better solution quality, reduced error rate in planning and fast time-to-market for a product. The literature survey highlights the research gaps, where the existing integrated ASP and ALB optimisation is limited to a Genetic Algorithm (GA) based approach, while Particle Swarm Optimisation (PSO) demonstrates better performance in individual ASP and ALB optimisation compared to GA. In addition, the existing works are limited to simple assembly line problems which run a homogeneous model on an assembly line. The aim of this research is to establish a methodology and algorithm for integrating ASP and ALB optimisation using Particle Swarm Optimisation. This research extends the problem type to integrated mixed-model ASP and ALB in order to generalise the problem. This research proposes Multi-Objective Discrete Particle Swarm Optimisation (MODPSO), to optimise integrated ASP and ALB. The MODPSO uses the Pareto-based approach to deal with the multi-objective problem and adopts a discrete procedure instead of standard mathematical operators to update its position and velocity. The MODPSO algorithm is tested with a wide range of problem difficulties for integrated single-model and mixed-model ASP and ALB problems. In order to supply sufficient test problems that cover a range of problem difficulties, a tuneable test problem generator is developed. Statistical tests on the algorithms' performance indicates that the proposed MODPSO algorithm presents significant improvement in terms of larger non-dominated solution numbers in Pareto optimal, compared to comparable algorithms including GA based algorithms in both single-model and mixed-model ASP and ALB problems. The performance of the MODPSO algorithm is finally validated using artificial problems from the literature and real-world problems from assembly products.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my appreciation and gratitude to my supervisor, Professor Ashutosh Tiwari, who have supported me from day one with his patience, guidance and knowledge. Special thanks go to Mr. Windo Hutabarat for his encouragement and effort. Without them, this thesis would not have been completed.

I also would like to thank my sponsors, the Ministry of Higher Education, Malaysia and Universiti Malaysia Pahang for giving me this opportunity. This acknowledgment is also dedicated to Cranfield University which has provided the best possible support during the research period.

Special thanks go to all my research colleagues, especially in Building 50, for their helpfulness and friendship.

Finally, my sincere thanks go to my family; my beloved mother who continuously prays for my success; my wife, Dini for her patience and understanding; and my children, Fahim, Fitry and Dhiya, who are my motivation for accomplishing this thesis. May God reward them with goodness.

LIST OF PUBLICATIONS

Published Conference Paper

1. Rashid, M., Tiwari, A. And Hutabarat, W. (2011), “An Integrated Representation Scheme for Assembly Sequence Planning and Assembly Line Balancing”, Proceedings of the 9th International Conference of Manufacturing Research (ICMR 2011), pp. 125-131, 6-8 September 2011, Glasgow, UK.

Published Journal Papers

1. Rashid, M., Hutabarat, W. and Tiwari, A. (2012), “A Review on Assembly Sequence Planning and Assembly Line Balancing Optimisation using Soft Computing Approaches”, *International Journal of Advanced Manufacturing Technology*, vol. 59, no. 1-4, pp. 335-349.
2. Rashid, M., Hutabarat, W. and Tiwari, A. (2012), “Development of Tuneable Test Problem Generator for Assembly Sequence Planning and Assembly Line Balancing”, *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 226, no. 11, pp.1900-1913.

Submitted Journal Papers

1. Rashid, M., Hutabarat, W. and Tiwari, A. (2012), “Multi-Objective Discrete Particle Swarm Optimisation Algorithm for Integrated Assembly Sequence Planning and Assembly Line Balancing”, *Journal of Applied Soft Computing*.
2. Rashid, M., Tiwari, A. and Hutabarat, W. (2013), ”Optimisation of Integrated Mixed-Model Assembly Sequence Planning and Assembly Line Balancing using Particle Swarm Optimisation”, *International Journal of Advanced Manufacturing Technology*.
3. Rashid, M., Tiwari, A. and Hutabarat, W. (2013),”Optimisation of Real-World Integrated ASP and ALB Problem using Multi-objective Discrete Particle Swarm Algorithm”, *Engineering Optimisation*.

4. Rashid, M., Hutabarat, W, and Tiwari, A. (2013), “Comparison of Integrated and Sequential Optimisation Approaches for ASP and ALB”, *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture.*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF COMMONLY USED ABBREVIATIONS

ACO	Ant Colony Optimisation
ALB	Assembly Line Balancing
ASP	Assembly Sequence Planning
ct	Cycle time
ct_{max}	Maximum cycle time
D	Assembly direction
DPSO	Discrete Particle Swarm Optimisation
ER	Error ratio
FR	Frequency ratio
GA	Genetic Algorithm
GALBP	Generalised Assembly Line Balancing Problem
GD	Generational distance
HGA	Hybrid Genetic Algorithm
HSD	Honestly Significant Different
M	Assembly time
MMALB	Mixed-Model Assembly Line Balancing
MODPSO	Multi-Objective Discrete Particle Swarm Optimisation
MOGA	Multi-Objective Genetic Algorithm
MOPSO	Multi-Objective Particle Swarm Optimisation
n	Number of assembly task
NSGA-II	Elitist Non-Dominated Sorting Genetic Algorithm
nws	Number of workstation
OS	Order Strength
PSO	Particle Swarm Optimisation
pt	Processing time
SA	Simulated Annealing
SALBP	Simple Assembly Line Balancing Problem
$Spread_{max}$	Maximum spread
T	Assembly tool
TPG	Test problem generator
TV	Time variability ratio
v	Workload variation
ws	Workstation
$\tilde{\eta}$	Number of non-dominated solution in Pareto optimal

CHAPTER 1

INTRODUCTION

The current global market continuously puts pressure on manufacturers to compete with competitors from all over the world. In order to ensure that their products remain competitive, manufacturers need to speed up the time-to-market and at the same time minimise manufacturing cost (Padrón et al., 2009). In addition, manufacturers also need to utilise all the resources at an optimum level (Amin and Karim, 2013).

Assembly is considered as one of the important processes in manufacturing. It consumes up to 50% of total production time and accounts for more than 20% of total manufacturing cost (Pan, 2005). Assembly is a sub-system of the manufacturing system and involves bringing and joining parts and/or sub-assemblies together (Marian, 2003). Regarding the challenge of remaining competitive in the global market, assembly optimisation activities are necessary to optimise the assembly resources. The concurrent assembly optimisation reduces the time-to-market for a product. This research details the integrated multi-objective optimisation of two assembly optimisation activities (i.e. Assembly Sequence Planning and Assembly Line Balancing) using Particle Swarm Optimisation algorithm.

1.1 Introduction to Assembly Optimisation

Assembly optimisation involves bringing and joining parts and/or sub-assemblies to make the process as efficient as possible (Rashid et al., 2012a). There exists a substantial amount of recent work on assembly optimisation. This work employs a variety of optimisation approaches. The research in assembly optimisation is classified according to the three stages of product development and production, as shown in Figure 1.1 (Marian, 2003).

The main assembly issue in Product Conception and Design stage is to apply Design for Assembly (DFA) methodology to reduce the number of parts and complexity in assembly. Besides reducing cost, DFA also brings additional benefits in terms of increased quality, reliability and shorter manufacturing time. The approach shortens the product cycle and ensures a smoother transition from prototype to production (Corallo et al., 2010). In general, any optimisation activities which involve the design of products are categorised as Product Conception and Design family.

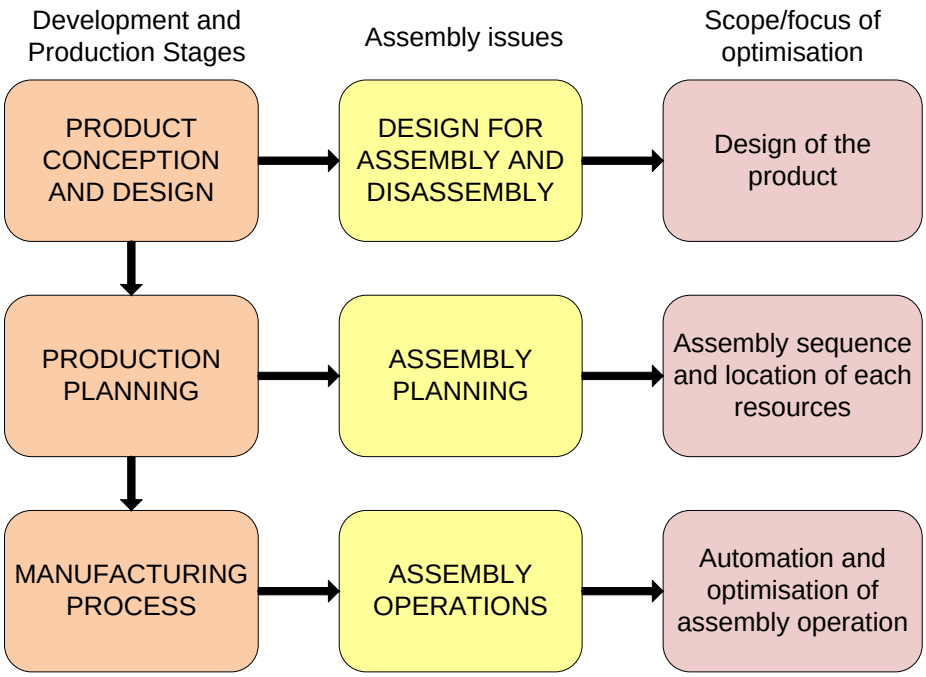


Figure 1.1: Assembly related issues in different product development stages (Marian, 2003)

Assembly optimisation in the Production Planning stage deals with the determination of optimum assembly sequence and the determination of optimum location of each resource. The best known optimisation activity in this stage is Assembly Sequence Planning (ASP), which has been studied since the 1980s. Solving the ASP problem is crucial because it determines many assembly aspects, including tool changes, fixture design and assembly freedom. Assembly sequence also influences overall productivity because it determines how efficiently and accurately the product is assembled.

During the Manufacturing Process stage, assembly optimisation focuses on two major activities. The first activity is determining the optimum automation level in assembly. The purpose of this activity is to apply the appropriate automation level in assembly in order to balance the investment in automation and the output. The second activity in this stage is assigning the assembly tasks into workstations, so that the workstations have equal or almost equal load (Marian, 2003). This activity is usually known as Assembly Line Balancing (ALB). In this stage, research in assembly optimisation focuses more on ALB problems rather than optimisation of automation levels. This can be observed through the number of publications as presented in Section 2.7.

Besides the straight-forward approach of optimising the assembly optimisation activities sequentially, researchers have considered integrating these activities. Many research works have been conducted designed to optimise the product design and ASP concurrently. For example, an integrated framework combining DFA and ASP has enabled the concurrent generation of preliminary design solution information and the assembly sequence information at the product design stage (Demoly et al., 2011). Many other works have also studied the integration of assembly optimisation within the Product Conception and Design stage and Production Planning stage (Pan et al., 2006; Demoly et al., 2012; Zha and Du, 2001).

However, research works studying the integration of assembly optimisation between the Production Planning stage and Manufacturing Process stage remain limited, as presented in Section 2.7.1. This research therefore focuses

on integrated optimisation of ASP and ALB activities which are classified in the Production Planning and Manufacturing Process stages respectively. In general, both ASP and ALB share important similarities, especially when focusing on increasing production with maximum resource utilisation. Both activities also share similar concepts such as assembly time and precedence constraint. ASP and ALB are both categorised as NP-hard problems where the solution space is increased excessively when the number of tasks are increased (Goldwasser and Motwani, 1997; Wee and Magazine, 1982). It makes the selection of appropriate optimisation algorithm crucial.

1.2 ASP and ALB

Assembly Sequence Planning (ASP) refers to a task for which planners, on the basis of their particular heuristics in assembling all the components of a product, arrange a specific assembly sequence according to the product design description (Tseng and Tang, 2006). Usually, the ASP research objective is to optimise the assembly sequence in terms of assembly time, assembly direction, tool changes and assembly stability (Hui et al., 2009; Gao et al., 2010; Wang and Liu, 2010). Figure 1.2 shows a common assembly representation using a Directed Acyclic Graph (DAG).

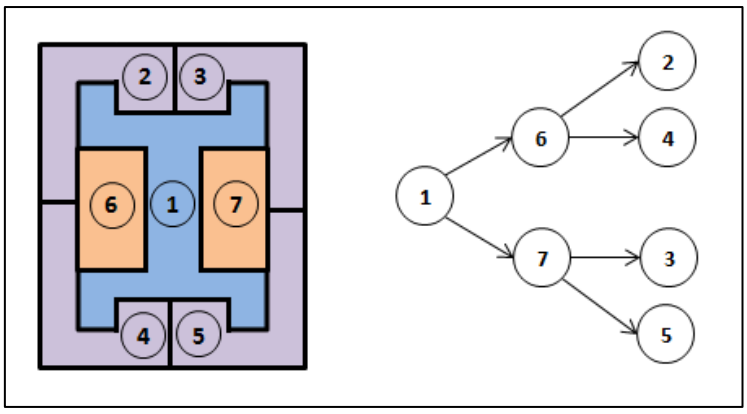


Figure 1.2: Assembly representation using Directed Acyclic Graph

From this graph, numerous feasible assembly sequences can be generated such as {1 6 7 2 4 3 5}, {1 7 3 6 5 4 2} or {1 6 2 7 4 3 5}. Based on this example

it can be seen that ASP is about determining the optimum sequence to assemble a product from all feasible assembly sequences.

An assembly line task involves the establishment of stations and products to be assembled (Tseng and Tang, 2006). According to researchers, Assembly Line Balancing (ALB) means the decision problem of optimally partitioning the assembly work among the stations with respect to particular objectives (Becker and Scholl, 2006).

For example in Figure 1.2, let the optimum assembly sequence from ASP be {1 6 2 7 4 3 5} and this assembly job will be assigned to three workstations. There are many possible assembly job assignment combinations, such as {(1 6), (2 7 4), (3 5)} or {(1 6 2), (7 4), (3 5)} or {(1 6 2 7), (4 3), (5)}. ALB determines the best assembly job combinations which feature equal or almost equal workload between workstations. In ALB, some of the optimisation objectives are to minimise the number of workstations, minimise the workload variance, minimise the idle time and maximise the line efficiency (Suwannarongsri and Puangdownreong, 2008).

1.3 Multi-objective PSO

In ASP and ALB optimisation literature, several objectives have been used to determine the optimum solution for the problem. When an optimisation problem involves more than one objective, this problem is known as a multi-objective optimisation problem (Deb, 2001). Traditionally, the simplest way to optimise a multi-objective problem is to bundle all the objectives into a single evaluation term using some kind of weighted assignment.This approach requires high-quality prior knowledge and experience regarding the importance of one objective compared to others.

Instead of focusing on one single optimum point, the researchers might be interested in all the best options available. There are many ways of defining a set of best options, but there is one predominant way, i.e. the Pareto optimal

solutions (Luke, 2010).In order to establish the set of “best option” solutions for multi-objective optimisation problem, the algorithm selection is critical.

The growth of heuristic algorithms has attracted many researchers to explore and apply these algorithms for multi-objective optimisation. One of the heuristic algorithms that have attracted researchers is Particle Swarm Optimisation (PSO). PSO is a population-based stochastic optimisation technique, developed by Kennedy and Eberhart in 1995. It was inspired by the social behaviour of bird-flocking or fish-schooling. PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA).

The system is initialised with a population of random solutions and searches for optimum solutions by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles (Kennedy and Eberhart, 1995). This is done by updating the particle position and velocity towards the current optimum solution.

The major advantage of PSO over the basic Evolutionary Algorithms (EAs), as highlighted by many researchers, is the simplicity of the algorithm (Shinzawa et al., 2007; Lu et al., 2010; Premalatha and Natarajan, 2009). The GA for example, requires 3 operations to converge, i.e. selection, crossover and mutation, while the PSO relies on velocity calculation to update the particle position (Rahmat-Samii, 2003). It reduces the computational time, as well as the memory usage.

Another PSO advantage is that it maintains the best solution history for an individual particle and also among the particles. Each particle remembers its previous velocity and the previous best position and uses them in its movement (Pasupuleti and Battiti, 2006). These features enable the PSO to maintain a balance between exploration and exploitation in the swarm and achieve fast convergence (Jeong et al., 2009).

In addition, the PSO algorithm is converged on the basis of “constructive cooperation” rather than “survival of the fittest” as in EAs (Shayeghi et al., 2010; Zeng and Jiang, 2010). This character ensures that all the particles in the initial population reach the final iteration (Sinha and Purkayastha, 2004). Therefore, by using PSO, better final solution variety can be achieved at the end of the optimisation process.

In addition to the advantages of PSO as discussed above, the PSO algorithm also proved to perform better than GA in ASP and ALB optimisations. In the majority of the ASP and ALB optimisations which compared the performance of GA and PSO, it was concluded that PSO has better overall performance than GA. The detail of the performance comparison between GA and PSO for ASP and ALB optimisation is presented in Section 2.4.5. Based on this fact, PSO is more promising to be used for ASP and ALB optimisation.

1.4 Research Problem and Motivation

Research works in individual ASP and ALB optimisation have seen rapid growth with hundreds of publications since the 1960s. However, only a limited amount of the research optimises both activities together. From the literature review in Section 2.7.1, only Genetic Algorithms have been used to optimise the integrated ASP and ALB, despite the fact that the PSO algorithm offers a good prospect based on its advantages and track record in individual ASP and ALB optimisation.

The assembly sequence plays an important role in the assembly plan. Many aspects of the assembly process, such as assembly line layout, assembly resource utilisation, etc., are designed and arranged by referring to the assembly sequence. In addition, good assembly sequences tend to improve the assembly efficiency and reduce the assembly cost (Wang and Liu, 2010). On the other hand, ALB also plays a vital function in assembly. The installation of an assembly line is a long-term decision and usually requires large capital

investments. Therefore, it is important that such a system is designed and balanced so that it works as efficiently as possible (Becker and Scholl, 2006).

In current practice, the ASP and ALB optimisation are performed sequentially. Normally the ASP is optimised before the ALB because it belongs to different product development stages. This practice causes a few problems since the ASP and ALB are interlinked. One such problem is that the sequential optimisation causes sub-optimal assembly operations, which means that the final solutions only fully satisfy one party (normally ASP). This problem occurs because of different search space sizes between ASP and ALB, as shown in Figure 1.3. In comparison with ASP search space, the search space of the ALB (a subsequent activity) is reduced because it is formed by the output of ASP optimisation.

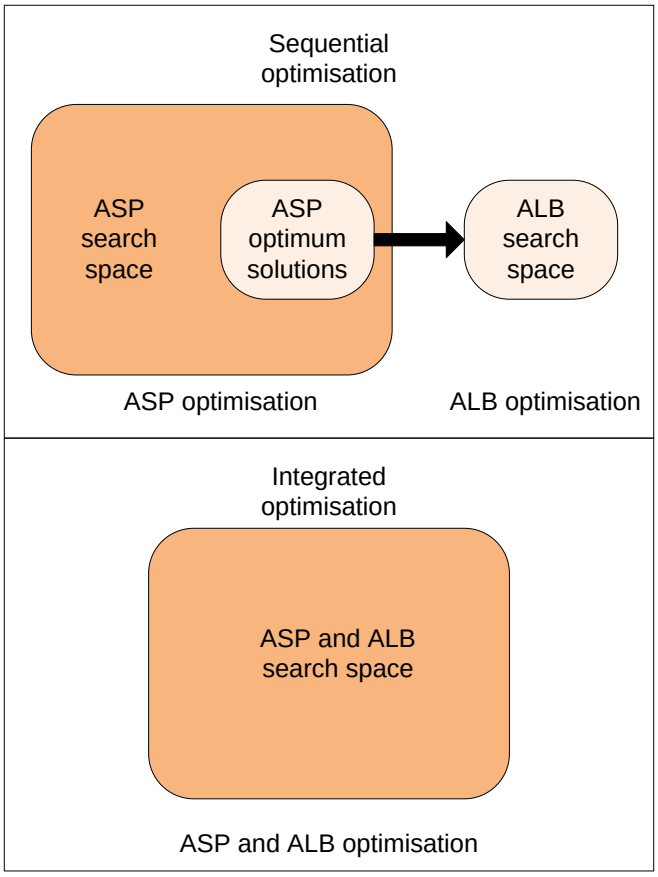


Figure 1.3: Search space different between sequential and integrated optimisation

Another problem which is caused by reduction of search space from ASP to ALB is the loss of possible optimum solutions. Since the solution space of ALB is filtered according to ASP objectives, there exists the possibility of losing the optimum solutions which fulfil the criteria for both activities. In this case, a better solution for ASP and ALB might be one of the solutions filtered away during ASP optimisation. This can be avoided by performing the integrated optimisation for ASP and ALB.

The integrated ASP and ALB problem is more challenging compared to individual ASP or ALB one, due to the complexity of the problem. The ASP and ALB problems individually are categorised as NP-hard combinatorial problems, where the solution spaces are excessively increased when the number of tasks increases (Goldwasser and Motwani, 1997; Wee and Magazine, 1982). When the optimisation of both activities is performed together, the problem difficulties are increased and require proper optimisation set-up including the algorithm selection. However, the integrated ASP and ALB is expected to create a better quality of assembly plans because of the provision of a larger search space for ALB compared to sequential optimisation.

1.5 Structure of the Thesis

This thesis is structured into nine chapters, as presented in Figure 1.4.

Chapter 1 gives an introduction to the research. It also presents the research problem and motivation.

Chapter 2 reviews literature in the ASP and ALB optimisation, including the individual optimisation, assembly problem types and optimisation algorithm. In this chapter, the literature survey is also performed to identify the research trends and research gaps in the area.

Chapter 3 outlines the research aim, objectives and scope. In addition, this chapter also presents the research methodology to present the overview of how this research is conducted.

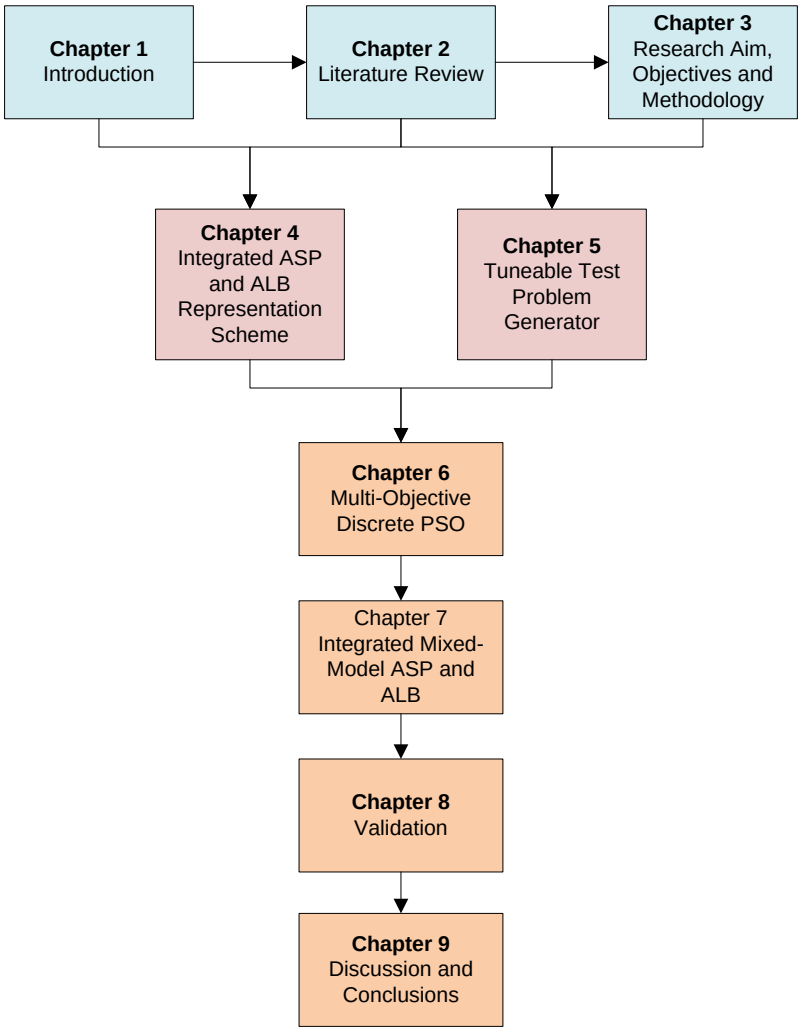


Figure 1.4: Thesis structure

Chapters 4 to 8 describe the main research activities and explain how research objectives are met. From Chapters 5 to 8, each chapter has its own numerical experiment and results to ensure validity, which form the basis for the following chapter.

Chapter 4 presents the representation scheme used to represent the integrated ASP and ALB problem. The proposed integrated representation will be the basis for optimisation in this research. In this chapter, an example based on assembly product is presented to show how the representation is established from a real product.

Chapter 5 proposes a tuneable test problem generator for integrated ASP and ALB with the purpose of generating sufficient test problems to cover a range of problem difficulties. This is important to overcome the limitation of test problems and also to ensure that the proposed algorithm is tested with a wide range of problem difficulties.

Chapter 6 presents the proposed algorithm to optimise integrated ASP and ALB problems. The algorithm called Multi-Objective Discrete Particle Swarm Optimisation (MODPSO) is specifically developed to deal with discrete problems as in ASP and ALB. The performance of the proposed MODPSO algorithm is then tested with the problems generated from the tuneable test problem generator.

Chapter 7 extends the application of the proposed MODPSO to optimise integrated mixed-model ASP and ALB. This assembly line is important to enhance the product variety using minimum investment cost. In this chapter, the formulation of integrated mixed-model ASP and ALB is explained. Comprehensive testing is also conducted to identify the ability of MODPSO to optimise this problem.

Chapter 8 validates the performance of the proposed MODPSO algorithm using problems from the literature. The optimisation results using the proposed MODPSO are compared with results presented in the literature. The MODPSO's performance is validated using real-world problems. Following that, a numerical comparison between integrated and sequential optimisation approaches is presented.

Chapter 9 discusses and concludes the contribution of the research findings to the knowledge and the limitations of this research. Finally, this chapter discusses the future direction of the research.

1.6 Chapter Summary

In summary, this chapter addresses the following points:

- The research in assembly optimisation, including Assembly Sequence Planning and Assembly Line Balancing has been introduced.
- Multi-objective optimisation and Particle Swarm Optimisation has been introduced.
- The research problem, motivation and challenge of the integrated ASP and ALB optimisation have been presented.
- The structure of this thesis has been explained.

CHAPTER 2

LITERATURE REVIEW

This chapter reviews the existing work on ASP and ALB optimisation which uses soft computing methods. A soft computing method is defined as an approach that is characterised by the use of inexact solutions to computationally-hard tasks for which an exact solution cannot be derived in polynomial time (Lendak et al., 2010). The aim of this chapter is to give an overview of ASP and ALB including the problem representation, constraints and optimisation objectives. This chapter also discusses the trends, potential and research gaps in ASP and ALB optimisation. This chapter attempts to achieve the following goals:

- Provide an overview of research in ASP and ALB optimisation.
- Identify the representation schemes used to represent ASP and ALB problems.
- Analyse the soft computing methods used to optimise ASP and ALB.
- Discuss the research trends and potential in ASP and ALB.
- Identify the research gaps in ASP and ALB optimisation.

2.1 Assembly Sequence Planning

Assembly Sequence Planning (ASP) is one of the most important components in assembly planning. ASP refers to a task for which planners, on the basis of their particular heuristics in assembling all the components of a product, arrange a specific assembly sequence according to the product design description (Tseng and Tang, 2006).

ASP is an NP-hard combinatorial problem where the solution space is increased excessively when the number of components is increased (Goldwasser and Motwani, 1997). Consider a product with six components which can be assembled in any sequence. In such a case, the number of possible solutions for this product is given by s=6!, which is equal to 720 solutions. When the number of components is increased to seven, the possible solutions for the products are increased to 7! = 5040. Additionally, in a real assembly problem, there are some constraints that need to be considered when generating assembly sequences.

Previous research shows that many approaches were proposed and used by researchers to represent the ASP problem. The most prominent way to represent it is by using the directed graph method, as used in a number of works (Qin and Xu, 2007; Sinanoğlu and Börklü, 2005; Chen et al., 2006; Chen et al., 2008). An assembly can be described by a directed graph $D = (P, C)$. P is a finite non-empty set of vertices, and C is a set of edges connecting them. Each vertex represents a component, and each edge represents a relationship between the two components. In some cases, the vertices and edges bring additional information such as assembly orientation, tool, assembly type and assembly time (Chen et al., 2002).

In assembly representation, the directed graph is specifically known as a ‘precedence diagram’, since the graph represents the precedence relation of assembly (Mitrović-Minić and Krishnamurti, 2006). Figure 2.1 shows the assembly precedence diagram with additional assembly information. In this diagram, the vertices represent the assembly components. Meanwhile, the

information {T_1, T_2, T_3 and T_4} represent the assembly tools and {+x,-x,+y,-y,+z,-z} represent the assembly direction for a particular component. Therefore, when an assembly sequence is established, it can be evaluated based on information in the assembly precedence diagram.

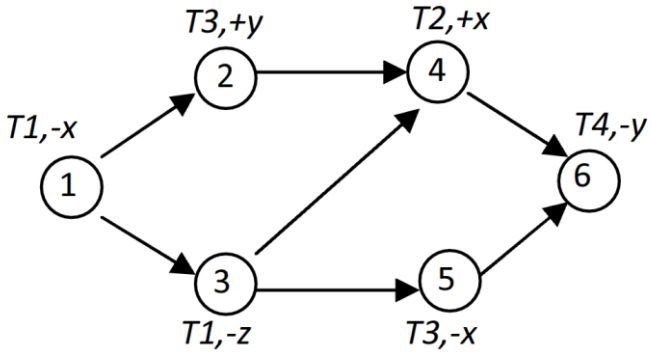


Figure 2.1: Assembly precedence diagram with additional information

The assembly sequence is evaluated according to some objective function. In previous research, to establish the objective function, the tool and direction variable is transformed to a measurable format such as cost, time or penalty index (Chen et al., 2008). As well as this approach, researchers also use a connector-based approach to represent ASP problem (Tseng and Tang, 2006; Wang and Tseng, 2009; Chang et al., 2009). In this method, each connector may assemble two or more components. Every task represents one connector in the assembly process and brings the information of fasteners' type, assembly direction, tools type and standard assembly time.

2.1.1 ASP Constraints

According to Marian (2003), there are two types of constraints in assembly, which are 'absolute constraints' and 'optimisation constraints'. The absolute constraints refer to constraints that, if violated, lead to infeasible assembly sequence. Meanwhile, the optimisation constraints are the constraints which lead to lower quality of assembly sequences when violated.

In the ASP context, the absolute constraints usually considered are precedence and geometrical constraints. Precedence constraint shows the relation of

predecessor and successor components for the assembly process. The precedence constraint cannot be violated, otherwise the infeasible assembly sequence will be generated. Precedence constraint can be represented in a precedence diagram (Figure 2.1) or in matrix form. Table 2.1 shows the precedence matrix for the precedence diagram in Figure 2.1. In this matrix, if part *i* must be assembled before part *j*, *P*(*i*, *j*) = 1. Otherwise, the matrix will be left empty.

Table 2.1: Precedence matrix (*P*) for Figure 2.1

i	*j*					
	1	2	3	4	5	6
1		1	1			
2				1		
3				1	1	
4						1
5						1
6						

Meanwhile, geometrical constraint in assembly concerns assembling the components without any collision. When mating two parts, there must be at least one collision-free trajectory that enables the components to be assembled correctly. All valid assembly sequences must meet geometric constraints for a given structure. Researchers use a matrix to describe geometric constraints between components in an assembly (Chen and Liu, 2001). For each pair of components (P_i, P_j), the matrix records directions in which P_i can be assembled without colliding with P_j. Then, a set of valid assembly directions for each (P_i, P_j) is defined, as the moving wedge (constraint that will guide to feasible sequences) of P_i with respect to P_j, denoted by $MW(P_i, P_j)$. They compute moving wedges for all pairs of components and store all moving wedges in the *MW* matrix.

On the other hand, the constraints classified as optimisation constraints are associated with the optimisation objectives of the problem. The constraints classified in this category include assembly tool constraint, assembly direction constraint and assembly stability constraint.

Researchers also use the tool matrix, TM = $[t_{ij}]_{nxm}$ to represent the tool constraint (Wang and Liu, 2010), where *n* is the number of parts and *m* is the number of practicable tools to assemble the corresponding part. After the optimum or near-optimum assembly sequences have been generated, the corresponding tools are also confirmed and at the same time, the number of changes (n_t) of the assembly tools can be obtained.

Meanwhile, the assembly direction constraint is represented as a penalty index in a number of different works (Gao et al., 2010; Wang and Tseng, 2009; Cao and Xiao, 2007). In these papers, when the subsequent assembly direction differs from the current direction, a penalty will be given to that particular assembly sequence.

The stability constraint is defined when the assembling parts maintain relative position and do not break contact during the assembly operation (Sinanoğlu and Börklü, 2005). Wang and Liu (2010) categorised the stability defined by the assembly connectors into three levels according to connection strength; i.e. strong, weak and unstable connection. This classification is also used by Yu et al. (2009). In this approach, strong connection will be assigned '0' index, weak connection '1' index and unstable connection '2' index.

2.1.2 ASP Optimisation Objectives

In previous works, various optimisation objectives and soft computing techniques have been used to optimise ASP problems. Figure 2.2 shows the frequency of ASP objectives in the cited papers from the year 2000 to the middle of 2013. The most popular ASP objective is to minimise the number of assembly direction changes. This objective is applied in 35 out of 58 cited research papers in ASP. In this objective, the assembly directions concerned are along the three principal axes (+*x*, -*x*; +*y*, -*y*; +*z*, -*z*). When the direction of the next assembly part is different to the current direction, a penalty is given according to the magnitude of direction change. The optimum sequence according to this objective will be minimum penalty caused by direction change.

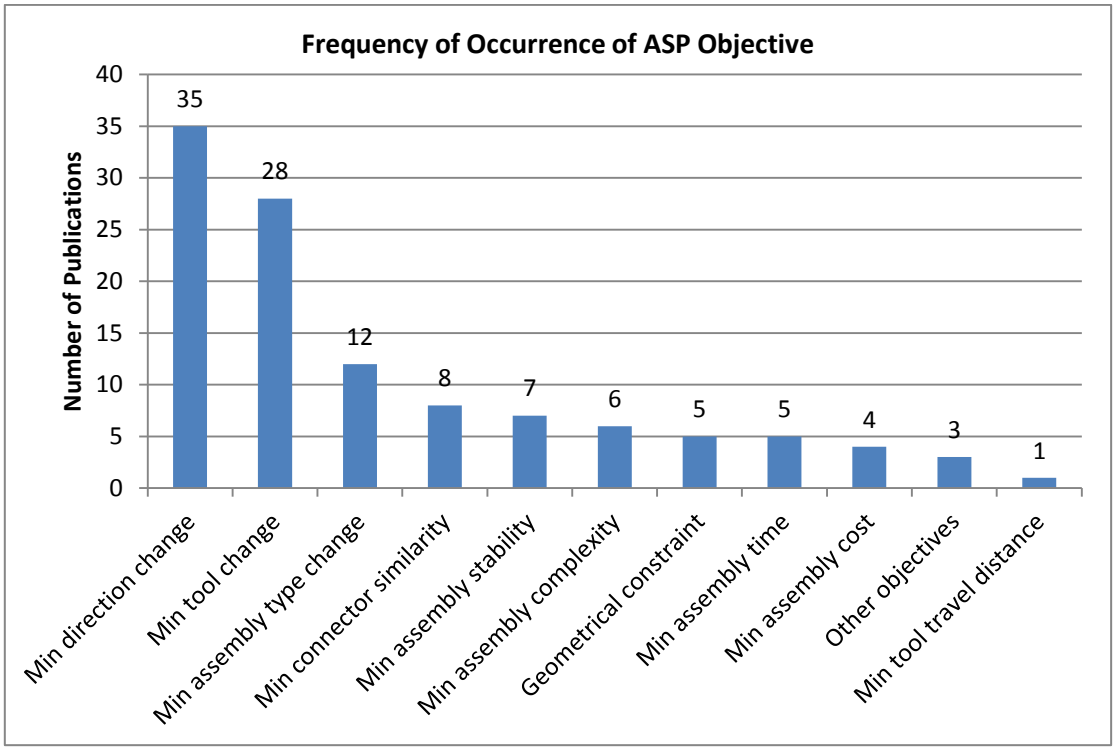


Figure 2.2: Frequency of occurrence of ASP objectives in cited research (2000-2013)

The second most popular ASP objective is to minimise the number of tool changes, which were used in 28 cited research papers. In assembly, tool change time is considered as non-productive activity and may create large down-time if not well-managed. In this case, when the next assembly process requires different tools from the current assembly process, the penalty will be given. The most optimum sequence following this objective is the sequence with the least tool change penalty.

As well as these two most frequent objectives, the ASP objective of minimising assembly type change is used in 12 cited research papers. This objective considers physical assembly features change such as mating, aligning, screwing, reverting, etc. (Lv and Lu, 2010; Lazzerini and Marcelloni, 2000; Guan et al., 2002; Lu et al., 2006).

Fourthly is the objective of minimising the connector similarity, whilst the objective of minimising assembly stability is in fifth place with 7 papers. This is

closely followed by the objective of minimising assembly complexity and the objective which is related to geometrical constraint. In ASP, some of the research stated that the geometrical constraint is a compulsory restriction. When the generated assembly sequence did not match geometrical constraint, the assembly sequence would not be evaluated. Therefore, this attribute is not included as an objective. However, in a significant amount of ASP research, the geometrical constraint is used as one of the ASP objectives (Yu et al., 2009; Li et al., 2003; Marian et al., 2006; Xing et al., 2010; Mingxing et al., 2011). When the assembly geometry is unfeasible, the large penalty index will be given to fitness function. Therefore, the unfeasible sequence will not appear as an optimum assembly sequence in the final results.

Next is the objective of minimising assembly time. In the ASP context, the objective of minimising assembly time is suitable to be applied in assembly cell type, where only one product is assembled in a single workstation at one time. In this case, the cycle time is equivalent to total processing time in the system. Figure 2.2 shows that the least frequent ASP objective used in the cited papers is to minimise assembly tool travel distance. This objective relates to the Printed Circuit Board (PCB) assemblies which involved robotic pick and place arms.

It must be borne in mind that more than half of the cited ASP research using multi-objective optimisation technique employed more than one objective in their research. Therefore, the total objectives frequencies, as shown in Figure 2.2, exceeded the total ASP cited research. Details regarding this information are available in Table 2.2.

2.2 Assembly Line Balancing

Assembly Line Balancing (ALB) is the decision problem of optimally partitioning the assembly work amongst the stations with respect to some objective (Chen et al., 2002). This problem aims at grouping assembly operations which have to be performed to produce final products, and assigning the groups of operations to stations, so that the total assembly time required at each station is

approximately the same and the precedence constraints between operations are respected (Gu et al., 2007).

In general, researchers divided ALB problem into two categories; Simple Assembly Line Balancing Problem (SALBP) and Generalised Assembly Line Balancing Problem (GALBP) (Scholl and Becker, 2006; Baybars, 1986; Boysen et al., 2007). SALBP deals with a serial assembly line which processes a unique model of a single product with all input parameters known with certainty (Betancourt, 2007). SALBP can be classified into three groups according to the objectives (Kilincci and Bayhan, 2006):

- SALBP-1: the objective is to minimise the number of stations on the line for a given cycle time.
- SALBP-2: the objective is to minimise the cycle time for a given number of stations on the line.
- SALBP-E: the objective is to maximise the line efficiency for variable cycle time and number of stations.

Meanwhile GALBP includes all of the problems that are not SALBP, such as balancing of mixed-model, parallel, U-shaped and two-sided lines with stochastic dependent processing times (Tasan and Tunali, 2006). In this section, only SALBP will be considered since it has accumulated a large number of works.

The simple ALB problem can be represented in a precedence diagram that contains n vertices and a set of edges. Each vertex represents an assembly task. Meanwhile, the vertices' weight shows the assembly time and the edges reflect the successor tasks.

Solving the ALB problem is about assigning the tasks $V_i(i = 1,2,...,n)$ into workstations W_j ($j = 1,2,...,m$) subjected to assembly constraints and optimisation objectives. In this problem, assembly time in each node is known as task time, t_i that refers to task i. Meanwhile, the total task time in workstation W_j is named as processing time, p_j. The highest processing time among all

workstations then is defined as cycle time, *ct*. In an assembly line, the cycle time will determine the production rate, *R*, which is given as follows:

$$R = \frac{1}{ct}$$

Eq. 2.1

Since cycle time is the highest processing time among all workstations, the difference between cycle time and processing time is unproductive time, which also known as idle time. For p_j is processing time in j^{th} workstation, the total idle time in assembly line is calculated as follows:

$$\text{Idle time} = m \cdot ct - \sum_{j=1}^{m} p_j$$

Eq. 2.2

As an example, the assembly tasks in Figure 2.3 are assigned to four workstations; W_1=[1,2], W_2=[3,5], W_3=[4,6] and W_4=[7,8]. It was found that the processing time for each workstation is p_1 = 7, p_2 = 10, p_3 = 10 and p_4 = 9. The highest processing time is found in W_2 and W_3, therefore the cycle time for this problem is c = 10 time units. The idle time for this solution is calculated as follows:

Idle time = 4(10) – (7+10+10+9)

= 4 time units

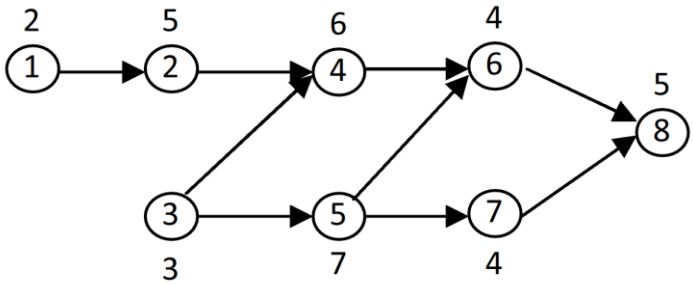


Figure 2.3: Precedence diagram for ALB

2.2.1 ALB Constraints

In ALB, the important constraints highlighted by researchers are occurrence constraint, precedence constraint and capacity constraint (Nof et al., 1997). The occurrence constraint refers to the restriction that ensures each task be assigned to exactly one workstation. For this purpose, an assignment matrix that consists of task and workstation variables is established. For i^{th} task and j^{th} workstation, x_{ij}=1 if task i is assigned to workstation j and 0 if otherwise. The precedence constraint was formulated as follows (Nof et al., 1997):

$$\sum_{j=1}^{m} j.x_{pj} - \sum_{j=1}^{m} j.x_{ij} \leq 0$$

Eq. 2.3

In this case, j refers to workstation number, m is total number of workstations, p is task/s that immediately precede task i. The x_{pj} = 1 if task p is assigned to workstation j and 0 if otherwise.

The capacity constraint depends on SALBP problem. For SALBP-1, the capacity constraint refers to the maximum allowable cycle time in the workstation. It can be formulated as follows:

$$\sum_{i=1}^{n} t_i \cdot x_{ij} \leq c$$

Eq. 2.4

In this equation, t_i refers to the processing time for task i and c is predetermined cycle time for the assembly line. Meanwhile, in SALBP-2, the capacity constraint is represented by the maximum number of workstations in the assembly line.

ALB research works have also addressed problems that consider additional restrictions apart from cycle time and precede constraints. For example, researchers considered a problem involving resource constraint, which defined the assembly space as one of constraint (Chica et al., 2010). Other examples

include zoning constraint and uniqueness constraint (Capacho and Pastor, 2006).

2.2.2 ALB Optimisation Objectives

Figure 2.4 shows the frequencies of ALB objectives that have been recorded from cited research papers.

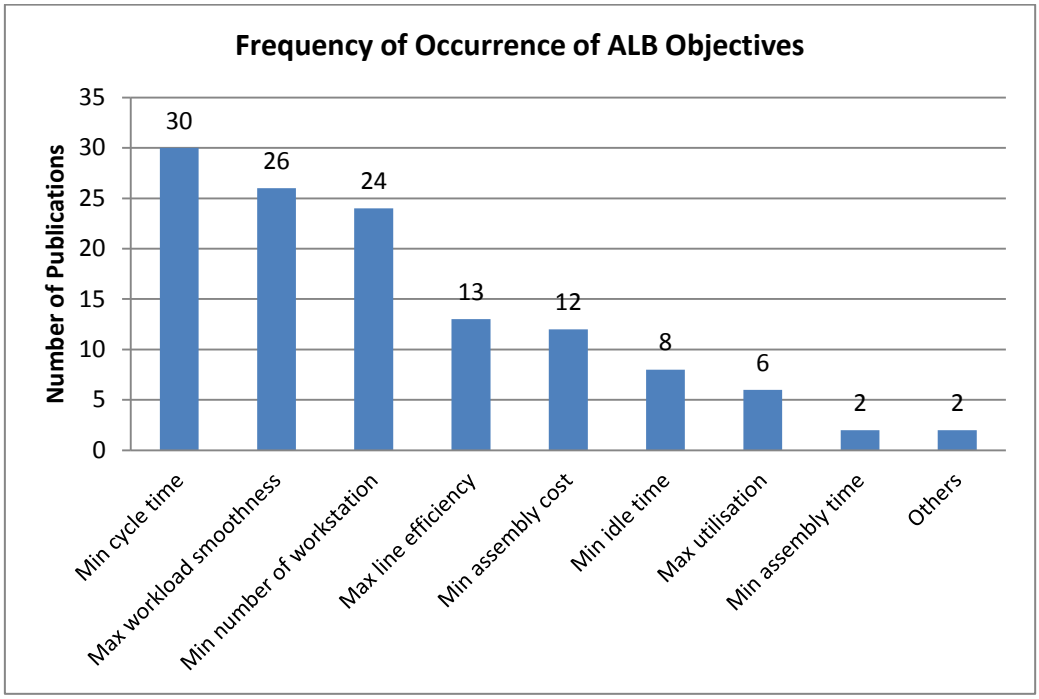


Figure 2.4: Frequency of occurrence of ALB objective in cited research (2000-2013)

The most frequent objective is to minimise cycle time which is recorded in 30 cited research papers. Cycle time is available time in each workstation to complete the required tasks to process a unit of product. It is also defined as the time interval between the processing of two consecutive units (Whitney, 2004). In ALB view, the cycle time is equal to longest processing time on any workstation.

Meanwhile, in second place is the objective of maximising workload smoothness, with 26 papers. In the assembly line, the basic workload smoothness is measured by calculating workload variation (v) as follows;

$$v = \frac{\sum_{i=1}^{nws} (ct\text{-}pt_i)}{nws}$$

Eq. 2.5

Where *ct* is cycle time, *pt* is processing time and *nws* is number of workstations. In this case, minimising the workload variation will maximise the workload smoothness (Chen et al., 2002).

The ALB objective of minimising the number of workstations is also important as it had been used in 24 cited papers. Usually, this objective is used in combination with upper limit of cycle time and another objective of maximising workload smoothness. In this case, the smallest number of workstations is not always the most optimum sequence because the workload balance will also need to be considered.

In ALB, the objective of maximising line efficiency has seen relatively moderate frequency of usage as it appears in 13 cited researches. Line efficiency is the ratio between total processing time in all workstations to the product of cycle time and number of workstation. The assembly line efficiency (*LE*) is given by the following equation;

$$LE = \frac{\sum_{i=1}^{nws} pt_i}{nws \times ct} \times 100$$

Eq. 2.6

where *nws* is number of workstations, pt_i is processing time in workstation i^{th} and *ct* is the cycle time.

The objective of minimising assembly cost is also moderately popular. To use this objective in ALB optimisation, many assumptions need to be made, such as labour cost, equipment utilisation cost and setup cost. Most of the related costs are dependent on variables like time, market price and geographical location. Therefore, this objective is only applicable to particular case studies.

The objective of minimising idle time and maximise utilisation in assembly have also been used in ALB optimisation. The idle time in each workstation is defined as the difference between processing time and allowed cycle time in assembly line (Zhang et al., 2009). This objective is indirectly adopted in workload variation, since the workload variation calculates the average idle time in each workstation. The assembly utilisation measure has been implemented in a variety of ways. McMullen and Tarasewich (2006) use assembly utilisation and associate the objective with labour utilisation. Meanwhile Chica et al. (2008), combine the labour and space utilisation to represent the assembly utilisation objective.

The least frequent objective designed to optimise ALB is to minimise the total assembly time. The total assembly time in the assembly line is defined as the total processing time in all workstations for a product. This objective is rarely used because in the ALB context, the cycle time is more important than the total processing time, since it will determine the production rate of the assembly line.

Table 2.2 shows the summary of the research in ASP and ALB using soft computing methods from January 2000 until middle 2013.

Table 2.2: Summary of literature in ASP and ALB using soft computing from 2000 to middle of 2013

Method	Author, Year	ASP	ALB	SO	MO (w)	MO (PO)	O 1	O 2	O 3	O 4	O 5	O 6	O 7	O 8	O 9	O 10	O 11	O 12	O 13	O 14	O 15	O 16	O 17
Genetic Algorithm (GA)	(Lazzerini and Marcelloni, 2000)	x			x		x	x	x														
	(Ponnambalam et al., 2000)		x			x											x	x	x	x			
	(Sabuncuoglu et al., 2000)		x		x													x	x				
	(Zhao and De Souza, 2000)		x	x													x						
	(Senin et al., 2000)	x		x												x							
	(Chen and Liu, 2001)	x		x				x															
	(De Lit et al., 2001)	x			x			x					x		x								
	(Smith and Liu, 2001)	x		x				x															
	(Gonçalves and De Almeida, 2002)		x	x																x			
	(Guan et al., 2002)	x			x		x	x	x														
	(Chen et al., 2002)	x	x			x	x										x	x					
	(Smith and Smith, 2002)	x		x				x															
	(Lin et al., 2003)	x				x	x	x															
	(Smith, 2004)	x		x				x															
	(Tseng et al., 2004)	x			x		x	x		x		x											
	(Bai et al., 2005)	x			x		x	x															
	(Udeshi and Tsui, 2005)	x		x							x												
	(Lu et al., 2006)	x				x	x	x	x														
	(Levitin et al., 2006)		x	x																x			
	(Marian et al., 2006)	x			x			x	x					x									
	(Pan et al., 2006)	x		x				x															

Method	Author, Year	ASP	ALB	SO	MO (w)	MO (PO)	O 1	O 2	O 3	O 4	O 5	O 6	O 7	O 8	O 9	O 10	O 11	O 12	O 13	O 14	O 15	O 16	O 17
	(Tseng and Tang, 2006)	x	x		x		x												x				
	(Tasan and Tunali, 2006)		x	x														x					
	(Gu et al., 2007)		x	x													x						
	(Qin and Xu, 2007)		x			x									x				x	x			
	(Zhang et al., 2007)		x		x												x						
	(Tseng et al., 2008)	x	x			x	x	x										x					
	(Zhang et al., 2008)		x			x									x		x	x					
	(Choi et al., 2009)	x			x										x	x							
	(Wang and Tseng, 2009)	x			x		x	x		x		x											
	(Tseng et al., 2010a)	x			x		x						x		x								
	(Yu and Yin, 2010)		x		x													x	x				
	(Mohd and Azmi, 2010)		x		x												x	x					
	(Razali and Geraghty, 2011)		x	x																	x		
	(Zeng et al., 2011)	x		x								x											
	(Zhou et al., 2011)	x			x		x	x															
	(Zacharia and Nearchou, 2012)		x		x													x			x		
	(Chen et al., 2012)		x	x														x					
	(Yolmeh and Kianfar, 2012)		x		x												x			x			
	(Wang et al., 2012)	x	x			x						x			x		x		x				
	(Kumar and Annamalai, 2012)	x		x												x							
	(Zhao et al., 2012)	x			x					x			x										
	(Hager et al., 2013)		x			x									x		x						
	(Mozdgir et al., 2013)		x	x														x					

Method	Author, Year	ASP	ALB	SO	MO (w)	MO (PO)	O 1	O 2	O 3	O 4	O 5	O 6	O 7	O 8	O 9	O 10	O 11	O 12	O 13	O 14	O 15	O 16	O 17
	(Mutlu et al., 2013)		x	x													x						
	(Perween et al., 2013)		x		x												x		x				
	(Zeng et al., 2013)	x			x		x	x	x														
Ant Colony Optimisation (ACO)	(McMullen and Tarasewich, 2003)		x		x										x		x					x	
	(Wang et al., 2005)	x		x				x															
	(Blum et al., 2006)		x	x															x				
	(McMullen and Tarasewich, 2006)		x		x										x		x					x	
	(Zhang et al., 2007)		x		x												x		x				
	(Blum et al., 2008)		x		x														x		x		
	(Chica et al., 2008)		x			x											x		x			x	
	(Zhang et al., 2008)		x							x					x	x							
	(Chen et al., 2008)	x		x						x													
	(Zhang et al., 2009)		x	x																	x		
	(Zhang et al., 2010)	x			x											x							
	(Chica et al., 2010)		x			x											x		x			x	
	(Chica et al., 2011)		x			x											x		x			x	
	(Sulaiman et al., 2011)		x	x													x		x				
	(Mingxing et al., 2011)	x			x								x	x									
	(Tseng, 2011)	x		x								x											
	(Zheng et al., 2012b)		x	x															x				
	(Zheng et al., 2012a)		x			x											x		x				
	(Mukund Nilakantan and Ponnambalam, 2012)		x		x												x					x	

Method	Author, Year	ASP	ALB	SO	MO (w)	MO (PO)	O 1	O 2	O 3	O 4	O 5	O 6	O 7	O 8	O 9	O 10	O 11	O 12	O 13	O 14	O 15	O 16	O 17
	(Chehade et al., 2013)		x		x										x					x			
	(Li et al., 2013b)	x		x			x																
	(Yu and Wang, 2013)	x					x	x					x										
Particle Swarm Optimisation (PSO)	(Liu et al., 2009)		x		x													x			x		
	(Yu et al., 2009)	x			x			x					x	x									
	(Lv and Lu, 2009)	x			x		x	x	x														
	(Wang and Liu, 2010)	x			x		x	x					x										
	(Xing et al., 2010)	x		x										x									
	(Lv and Lu, 2010)	x			x		x	x	x														
	(Nearchou, 2011)		x			x											x	x					
	(Lv, 2011)		x	x															x				
	(Akyol and Mirac Bayhan, 2011)		x		x												x	x					
	(Jianping et al., 2011)		x		x												x		x				
	(Petropoulos and Nearchou, 2011)		x														x	x					
	(Zhang and Huang, 2012)	x			x		x		x														
	(Zhu et al., 2012a)		x		x													x					x
	(Mukred et al., 2012)	x			x		x	x															
	(Raj et al., 2012)	x		x																			x
	(Liu and Wen, 2013)		x	x															x				
	(Li et al., 2013a)	x			x			x	x	x													
	(Muslim et al., 2013)	x		x												x							
	(Hamta et al., 2013)		x	x											x		x	x					

Method	Author, Year	ASP	ALB	SO	MO (w)	MO (PO)	O 1	O 2	O 3	O 4	O 5	O 6	O 7	O 8	O 9	O 10	O 11	O 12	O 13	O 14	O 15	O 16	O 17
Simulated Annealing (SA)	(Shan et al., 2006)	x			x		x	x	x														
	(Özcan and Toklu, 2009)		x		x													x		x			
	(Seyed-Alagheband et al., 2011)		x	x													x						
	(Cakir et al., 2011)		x			x									x			x					
	(Zhang et al., 2012a)	x		x																			x
	(Roshani et al., 2013)		x		x													x		x			x
GA+SA	(Qin and Xu, 2007)		x			x									x			x					
	(Li and Shan, 2008)	x			x		x	x	x														
	(Lin et al., 2009)		x			x									x	x							
	(Yuan et al., 2012)	x		x								x											
Petri Net	(Lapierre et al., 2006)		x	x																x			
	(Kilincci and Bayhan, 2006)		x	x																x			
	(Suwannarongsri et al., 2007)		x		x													x	x	x	x		
	(Kilincci, 2010)		x	x													x						
Memetic Algorithm	(Khoo and Alisantoso, 2003)		x		x						x										x		
	(Tseng et al., 2007)	x			x		x	x				x											
	(Chang et al., 2009)	x			x		x	x				x											
	(Liu et al., 2013)		x	x														x					
Immune Algorithm	(Liu et al., 2003)		x	x														x					
	(Cao and Xiao, 2007)	x			x		x	x															
	(Andrés et al., 2008)		x	x															x				
Imperialist Algorithm	(Zhou et al., 2012)	x			x		x	x															
PSO+SA	(Shan et al., 2009)	x			x		x	x	x														

Method	Author, Year	ASP	ALB	SO	MO (w)	MO (PO)	O 1	O 2	O 3	O 4	O 5	O 6	O 7	O 8	O 9	O 10	O 11	O 12	O 13	O 14	O 15	O 16	O 17
GA+SA+ACO	(Hui et al., 2009)	x		x						x													
Neural Network	(Kilincci and Bayhan, 2008)		x	x																x			
Tabu Search (TS)	(Suwannarongsri et al., 2008)		x		x													x	x	x	x		
GA+PSO	(Xing and Wang, 2012)	x																					x
GA+ TS	(Li et al., 2003)	x			x			x						x									
Other Heuristic Search	(Tijo and and Numar, 2008)		x	x															x				
	(Nearchou, 2008)		x			x											x	x					
	(Su, 2009)	x		x				x															
	(Yeh and Kao, 2009)		x	x															x				
	(Martino and Pastor, 2010)		x	x													x						
	(Zhang et al., 2012b)	x			x		x	x															

SO – Single Objective
MO (w) – Multi objective (Weighted-based)
MO (PO) – Multi objective (Pareto optimal)
O1 – Minimise tool change
O2 – Minimise assembly direction change/reorientation
O3 – Minimise assembly type change
O4 – Minimise assembly complexity
O5 – Minimise assembly tool travel distance
O6 – Minimise connector similarity
O7 – Maximise assembly stability
O8 – Geometrical constraint
O9 – Minimise assembly cost
O10 – Minimise assembly time
O11 – Minimise cycle time
O12 – Maximise workload smoothness
O13 – Minimise number of workstation
O14 – Maximise line efficiency
O15 – Minimise idle time
O16 – Maximise utilisation
O17 - Other

2.3 Representation Methods

In 1984, Bourjault presented a method to generate all feasible assembly sequences by introducing a question and answer approach (Bourjault, 1984). By using this method, $2l^2$ questions need to be answered where *l* is number of liaison or relation in specific assembly operations. Later, De Fazio and Whitney improved Bourjault's work by reducing the number of questions to 2*l* (De Fazio and Whitney, 1987). Then, Homem de Mello and Sanderson applied the AND/OR graph which provided a compact representation of all assembly sequences (Homem de Mello and Sanderson, 1990). This approach enhances the possibility of parallel execution of assembly operations. In Homem de Mello and Sanderson (1990), the directed graph representation is used to represent the ASP problem. This graph shows the precedence relation of the assembled parts. Qian et al. (1996) improved the idea of adopting procedural network in the data structure to represent assembly sequences. This approach is similar to the AND/OR graph, except the parts only appear once in the graph.

An assembly sequence graph is also used to represent the assembly sequence problem (Gottipolu and Ghosh, 1997). This graph uses nodes which are represented by blank or hatch boxes that imply assembly status. Lai and Huang (2004) applied an integrated framework of part liaison matrix and precedence Boolean relation to develop a systematic and integrated method of determining the assembly sequence planning of a product. Gu and Liu (2008) presented a symbolic ordered binary decision diagram (OBDD) with the purpose of minimising information storage space.

Most of the published work has applied a precedence graph to represent the ALB problem. A precedence graph is a type of acyclic directed graph specifically used to represent relations between assembly tasks. Some researchers also explore different representation strategies other than precedence graphs for ALB problems. Koç (2005) proposed the ALB

representation using the AND/OR graph with small changes to focus on assembly relations rather than parts.

Another proposed alternative approach is to use a sub-graph in assembly (Capacho and Pastor, 2006). This graph shows alternative assembly processes that involve different and independent sets of tasks. However, this approach does not show all possible assembly sequences in one graph. Salum and Supciller (2008) used the rule-based representation that applied '*if-then*' rules to determine feasible assembly sequence. Although there are a few researchers who try to use other representation approaches than precedence graph, the success of these approaches is not well established due to the limited number of further works and publications.

Besides independent representation for ASP and ALB problems, a small number of researchers had also performed an integration of ASP and ALB optimisations using different representation schemes. In 2002, Chen et al. used assembly task representation to present ASP and ALB data and constraints. However, for ASP problems they only consider minimising the number of tool changes as an optimisation objective. Since assembly tool is dependent on assembly task, it can be easily adopted in assembly task-based representation. Later, Tseng and Tang (2006) used a connector-based representation approach to present ASP and ALB problems. In this approach, all the assembly components are classified according to assembly connectors.

In summary, previous researchers had successfully developed various ASP representation schemes to fit with their particular problem characteristics and attributes. One of the common similarities among these schemes is that they are based on assembly parts. On the other hand, the most dominant and successful representation method in ALB is precedence graph, which is based on assembly task. In order to realise an integrated representation for both problems, it must be built using a similar basis. In this case, the assembly task basis is chosen after considering the available alternatives, flexibilities and success of previous works. Although there is a work that uses a similar approach (i.e. Chen et al., 2002), this work does not consider one of important

and frequently used optimisation objectives from previous survey: to minimise assembly direction change.

2.4 Optimisation Methods

Previous research in ASP and ALB optimisation shows that various soft computing methods were used. Figure 2.5 shows the number of papers which used different soft computing methods to optimise ASP and ALB problems from the year 2000. According to the diagram, the three most dominant optimisation methods, used in 72% of the cited research, are Genetic Algorithm (GA), Ant Colony Optimisation (ACO) and Particle Swarm Optimisation (PSO).

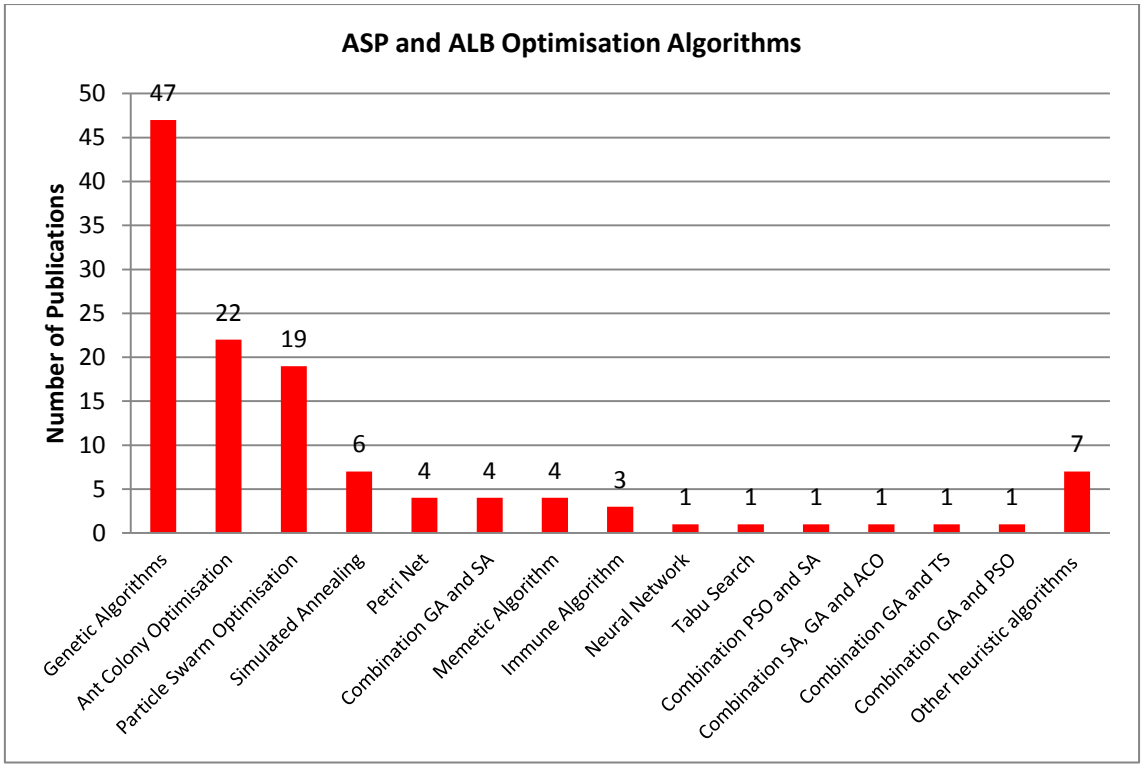


Figure 2.5: Number of papers that used different soft computing methods

2.4.1 Genetic Algorithms

Genetic Algorithm (GA) is inspired by evolutionary processes that are based on natural evolution. It was introduced by John Holland in 1975. This technique imitates the biological evolution theory, where by the concept of '*survival of the*

fittest' exists. GA provides a method of searching which does not need to explore every possible solution in the feasible region to obtain a good result (Goldberg, 2007).

In ASP and ALB optimisation, 47 out of 122 cited published works used this algorithm to find optimum assembly sequence. Researchers used GA in ASP problems because it can generate optimum or near-optimum solutions faster than exact algorithms (De Lit et al., 2001; Smith and Liu, 2001). In GA, the number of considered solutions is reduced compared to exact algorithms. Researchers also prefer to use GA in ASP and ALB because it can handle complex and multiple constraints problems well (Chen and Liu, 2001). Other than that, researchers were influenced by the success of GA in solving a wide variety of problems (Gu et al., 2007; Yu and Yin, 2010).

Although numerous papers exist which used this algorithm, GA in an original and basic form is unsuitable to be used directly to solve and optimise ASP and ALB problems. The first reason for this is that the original binary strings in chromosomes are less suitable for complex combinatorial problem such as ASP (Tasan and Tunali, 2008). The second reason concerns the feasibility of chromosomes in handling assembly precedence constraints. The GA in basic form tends to generate unfeasible offspring that violates precedence constraint because of crossover and mutation operators (Marian, 2003). To handle this constraint, researchers used different approaches like penalty and repair strategy.

In a number of works, the penalty approach was used to handle precedence constraint (Lu et al., 2006; Smith and Smith, 2002; Zhang et al., 2007). A penalty is given to chromosomes that are unfeasible due to violation of precedence constraints, resulting in reduced fitness. Therefore, the chance of unfeasible chromosomes being selected in the next generation is reduced. In addition, repair strategy is used in the literature to handle precedence constraint (De Lit et al., 2001; Gao et al., 2009). In this approach, the unfeasible chromosome is repaired and transformed into feasible chromosome using an additional step in GA. Other than that, a topological sort concept which

originated from graph theory is also applied in handling precedence constraint (Zacharia and Nearchou, 2012).

Even though GA has been successfully implemented in ASP and ALB, researchers have highlighted some issues regarding this algorithm. The main issue is that standard GA is susceptible to early convergence (Shan et al., 2006; Li and Shan, 2008). A further common issue concerns high computational time (Moon et al., 2009). Researchers have proposed an adaptive GA to solve premature convergence and high computational time issues (Yu and Yin, 2010; Moon et al., 2009). In adaptive GA, dynamic probabilities for crossover and mutation operators are introduced to vary computational time and selection rate. Another method of reducing computational time is performed by introducing Dynamic Partitioning (DPa) in the chromosome (Sabuncuoglu et al., 2000). DPa modifies chromosome structures by defining frozen and unfrozen task allocation in workstations for ALB. The task allocation is only made for unfrozen tasks, whereby the frozen task will remain unchanged as in the previous generation. Therefore, the computational time of this problem is reduced because of the lesser length of active chromosomes. Besides improving basic GA operators, researchers have combined GA with other soft computing algorithms to improve its performance. In general, combination of GA with Simulated Annealing, Tabu Search and Ant Colony Optimisation has resulted in better performance compared to the original GA (Shan et al., 2006; Li and Shan, 2008).

2.4.2 Ant Colony Optimisation

Ant Colony Optimisation (ACO) was introduced by Marco Dorigo in 1992 (Zhang et al., 2009). It is inspired by the pheromone trail-laying behaviour of real ant colonies. In ACO, a set of agents called artificial ants search for effective solutions to a given optimisation problem. To apply ACO, the optimisation problem is transformed into the problem of finding the best path on a weighted graph. The artificial ants incrementally build solutions by moving on the graph. The solution construction process is stochastic and is biased by a pheromone model, that is, a set of parameters associated with graph

components (either nodes or edges) whose values are modified at runtime by the ants.

ACO has attracted 22 publications in ASP and ALB in the past thirteen years due to various reasons. Researchers used this algorithm to overcome a shortcoming of GA that depends highly on initial chromosomes (Wang et al., 2005; Shuang et al., 2008). Besides that, the success of this algorithm in solving popular discrete problems such as Travelling Salesman Problem, machine scheduling problem and Vehicle Routing Problem have also inspired researchers to use ACO in ASP and ALB (Chica et al., 2008; Zhang et al., 2008). Another reason for ACO implementation is that ASP and ALB problems can be directly represented by a completed graph as in ACO (Wang et al., 2005).

In original ACO, one of the common drawbacks stressed by researchers is regarding the positive feedback system which only accumulates good solutions. In original ACO, the better the solution, the greater amount of pheromone deposited. However, the pheromone trail for all paths is set to be evaporated when it generates a bad solution. However, over-emphasis of this rule will cause premature convergence (Shuang et al., 2008). Meanwhile, Zhang et al. (2008) have proposed re-evaluating the unfit solutions, because they might be just a few iterations away from the global optimum.

The main focus of researchers improving the ACO algorithm is solving premature convergence. Zhang et al. (2008) have introduced a summation rule to replace the original pheromone “drop and evaporate” updating rule. In the pheromone summation updating rule, the best trail is determined by summation of total pheromone dropped without considering the evaporation factor. Meanwhile, another work adopted Particle Swarm position updating approach to overcome premature convergence in ACO (Shuang et al., 2008). The hybridisation of ACO and Particle Swarm not only solves the premature convergence in ACO, but also reduces computational time compared to the original ACO.

2.4.3 Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) was originated by Kennedy and Eberhart in 1995 (Kennedy and Eberhart, 1995). It is inspired by the social behaviour of bird flocking or fish schooling. The PSO is quite similar to GA, in which the system is initialised with a population of random solutions. However, unlike the GA, the PSO has no evolution operators such as Crossover and Mutation. The potential solutions, called particles, fly through the problem space by following the current optimum particles (Sinavandam and and Deepa, 2008).

In ASP and ALB, 19 papers applied this algorithm. Of these, 12 papers used PSO to solve multi-objective problems, but only four used the Pareto optimal approach to deal with multi-objective optimisation. Most of the researchers used traditional weighted approaches to solve multi-objective optimisation.

The PSO is a relatively new algorithm compared to GA and ACO. Not many papers which applied PSO to ASP and ALB were published before 2009. This has motivated researchers to apply PSO to ASP and ALB optimisation (Lv and Lu, 2010; Yu et al., 2009; Nearchou, 2011). Moreover, PSO is a simple algorithm because it only uses a single velocity formula to evolve (Lv et al., 2010). Therefore, PSO algorithm is easy to implement and requires fewer computational resources than GA.

However, similar to GA, the original PSO is not suitable to be directly applied to ASP and ALB problems. Besides the precedence constraint issue, the original PSO is designed for continuous problems, where the solution is in real-valued space, while ASP and ALB solutions reside in discrete integer space (Lv and Lu, 2010; Wang and Liu, 2010). Another important issue with original PSO is that it is easily trapped in local optimum (Wang and Liu, 2010). To solve this problem, researchers introduced a new mechanism of updating velocity by using one of two formulae randomly instead of single formula (Yu et al., 2009). Meanwhile, Wang and Liu (2010) introduced a chaotic operator to diversify the updated particle position, which finally helped to reduce premature convergence.

2.4.4 Other Methods

Besides the three main algorithms mentioned above, the researchers in ASP and ALB also used other soft computing methods, such as Simulated Annealing, Petri Net, Memetic Algorithm, Immune Algorithm, Neural Network and Tabu Search (Shan et al., 2006; Özcan and Toklu, 2009; Cakir et al., 2010; Suwannarongsri et al., 2007; Lapierre et al., 2006; Kilincci, 2010; Suwannarongsri and Puangdownreong, 2008; Chang et al., 2009; Khoo and Alisantoso, 2003; Tseng et al., 2007; Cao and Xiao, 2007; Liu et al., 2003; Andrés et al., 2008; Kilincci and Bayhan, 2008).

In addition, some researchers also combined soft computing methods to solve the ASP and ALB. Researchers combined GA with Simulated Annealing method and called them Genetic Simulated Annealing Algorithm (GSAA) (Qin and Xu, 2007; Li and Shan, 2008; Lin et al., 2009). Shan et al. (2009) combined PSO and Simulated Annealing to solve multi-objective ASP problems using the Pareto approach. The other algorithm combinations found to optimise ASP and ALB problems are GA and Tabu Search and GA, ACO and Simulated Annealing (Hui et al., 2009).

2.4.5 Comparison of GA, ACO and PSO Performance

GA, ACO and PSO have successfully been implemented to optimise problems in different areas. In Figure 2.5 these three algorithms accumulated more than 72% of the total number of cited papers. Researchers have conducted performance comparison tests among these algorithms to identify which algorithm demonstrates better performance in optimising a particular problem. Elbeltagi et al. (2005) for example compared five algorithms, including GA, ACO and PSO to optimise discrete problems. The optimisation result reported that the PSO was generally found to perform better than GA and ACO in success rate and solution quality, while GA leads in term of computational time.

GA, ACO and PSO algorithms have also been compared for flow shop scheduling (Abraham et al., 2008). From the tests performed, ACO and PSO

exhibited better make span for smaller instances compared to GA, but for larger instances, the PSO has out-performed both ACO and GA. Furthermore, PSO has better computational time in both small and large instances problems.

In ASP and ALB optimisation, although no cited papers compared the performance of these three algorithms, examples within the literature have compared GA and PSO performance. Lin et al. (2009) compared Multi-Objective PSO (MPSO) with Multi-Objective GA (MOGA) for incorporated ASP and supplier selection. The result indicated that the MPSO not only has better convergence but also obtained better Pareto optimal compared to MOGA. MOGA however, has better computational time in this application (Lin et al., 2009).

Xing and Wang (2012) have proposed the improved version of PSO by hybridising it with GA. Prior to the improved version, they have compared the performance of PSO and GA for ASP. Although the expected solution was not achieved by both algorithms, the PSO shows better convergence and overall performance with GA (Xing and Wang, 2012).

In ALB, Liu and Wen (2013) have proposed Multi-Objective Culture PSO (MCPSO) and compared it with Elitist Non-Dominated Sorting GA (NSGA-II), Improved Strength Pareto Evolutionary Algorithm (SPEA2) and Pareto Archived Evolution Strategy (PAES). The overall performance concluded that the MCPSO has superior performance compared to NSGA-II, SPEA2 and PAES (Liu and Wen, 2013).

The ASP and ALB discussed above are some of the works which compare PSO and GA performance. The list of published ASP and ALB works which compare PSO and GA performance is summarised in Table 2.3. Based on Table 2.3, the PSO is found to have better overall performance than GA in individual ASP and ALB optimisation. One of interesting trends found is that the PSO researchers tend to compare their algorithms with GA, but the GA researchers prefer to compare their algorithms within the GA community instead of relatively new algorithms like PSO and ACO.

Table 2.3: List of GA and PSO comparison papers for ASP and ALB

Problem	Author/s	Year	Better algorithm	
			Computational time	Overall performance
ASP	Lin et al.	2009	GA	PSO
	Lv et al.	2010	n/a	PSO
	Xing and Wang	2012	PSO	n/a
	Li et al.	2013	n/a	PSO
ALB	Rahimi-Vahed et al.	2007	n/a	PSO
	Kuo and Yang	2011	n/a	PSO
	Jianping et al.	2011	PSO	PSO
	Nearchou	2011	n/a	PSO
	Nilakantan and Ponnambalam	2012	n/a	PSO
	Hamta et al.	2013	PSO	PSO
	Liu and Wen	2013	PSO	PSO

2.5 Test Problems for ASP and ALB

In any optimisation algorithm development work, assessment of algorithm performance is the key to measuring the success or failure of particular algorithms. In ASP and ALB optimisation work focusing on algorithm development or improvement, researchers have used two approaches to test algorithm performance. One approach is to test the algorithms using specific case studies (Marian et al., 2006; Chica et al., 2010). Another acknowledged approach is to adopt the test problems frequently used in the literature (Kilincci and Bayhan, 2006; Smith, 2004). These approaches lack generality because there has been no investigation into the fit of algorithms to problem types. Algorithms have not been tested with a wide range of problem types.

The most frequently used test problem in ASP is an assembly of transmission-type parts with eleven components, presented by DeFazio and Whitney (De Fazio and Whitney, 1987). This problem has been presented in many papers to evaluate algorithm performance (Chen and Liu, 2001; Smith and Liu, 2001; Wang et al., 2005). Other than this widely-used problem example, most ASP

test problems found in the literature have only been used within the same research group. Hence, there is no accepted standard ASP test problem for evaluating algorithm performance. On the other hand, in ALB optimisation, development of test problems was begun in the 1960s, resulting in many test problems which have been developed and collected by different researchers. These problems vary in task size from eight to 297 tasks. The famous ALB problems such as the 8-tasks by Bowman, 45-tasks by Kilbridge and Wester, 70-tasks by Tonge, 111-tasks by Arcus and 297-tasks by Scholl are still being used today to evaluate algorithm performance for line balancing problems (Scholl, 1993).

Although these benchmark ASP and ALB problems are available for comparing algorithm performance, no standard test problem set exists that covers a wide variety of problem difficulties, especially to test the integrated ASP and ALB optimisation. Not only is this important for enhancing the researchers' understanding of their algorithm, it will also help users in selecting which algorithm is more appropriate to their requirements. In order to facilitate such experimentation, a set of problems with controllable complexity level is needed. One way to address this is to devise a test problem generator with tuneable difficulty level that can systematically generate a set of test problems with a desired mix of complexity levels.

2.6 Mixed-Model Assembly Problems

In general, the assembly line problems are classified into two categories, i.e. simple and generalised assembly line problems (Scholl and Becker, 2006). Simple assembly line only runs one homogenous product, on serial line layout and all workstations are equally equipped with machines and workers (Scholl and Becker, 2006). Meanwhile, the generalised assembly line includes all the problems that are not simple assembly problems, such as U-shape, two-sided lines, mixed-model and multi-model assembly line (Tasan and Tunali, 2008).

Beside the single-model assembly problems presented earlier, researchers also explored the mixed-model assembly problems. The mixed-model assembly line runs different product models in arbitrarily intermixed sequence on a single assembly line (Scholl and Becker, 2006). This type of assembly line is widely used in various industries to produce a variety of products on one single assembly line (Zhu et al., 2012b). The mixed-model assembly line is important in industry because sharing the different models of products on the same assembly line can save investment cost (Hu et al., 2008). Other than that, the mixed-model assembly line can also absorb the fluctuation of demand of different models using an assembly line (Hu et al., 2008).

Since the mixed-model assembly problem arises in the Manufacturing Process stage, the ASP that belongs to Production Planning stage is unrelated to this assembly type (refer to Figure 1.1). Therefore, no prior work on mixed-model ASP is to be found. On the other hand, a large amount of mixed-model ALB work is available.

The mixed-model and multi-model assembly problem can be distinguished according to the number of lot size. The multi-model line runs the product in batches, whereas the mixed-model line runs the lot size equal to one (Fokkert and Kok, 1997). In addition, the mixed-model produced the different models in arbitrarily intermixed sequence, while the multi-model runs a group of similar models with intermediate setup operation (Becker and Scholl, 2006). The graphical comparison between single-model, mixed-model and multi-model assembly line is presented in Figure 2.6. In this figure, different shapes represent different models that run on the assembly line.

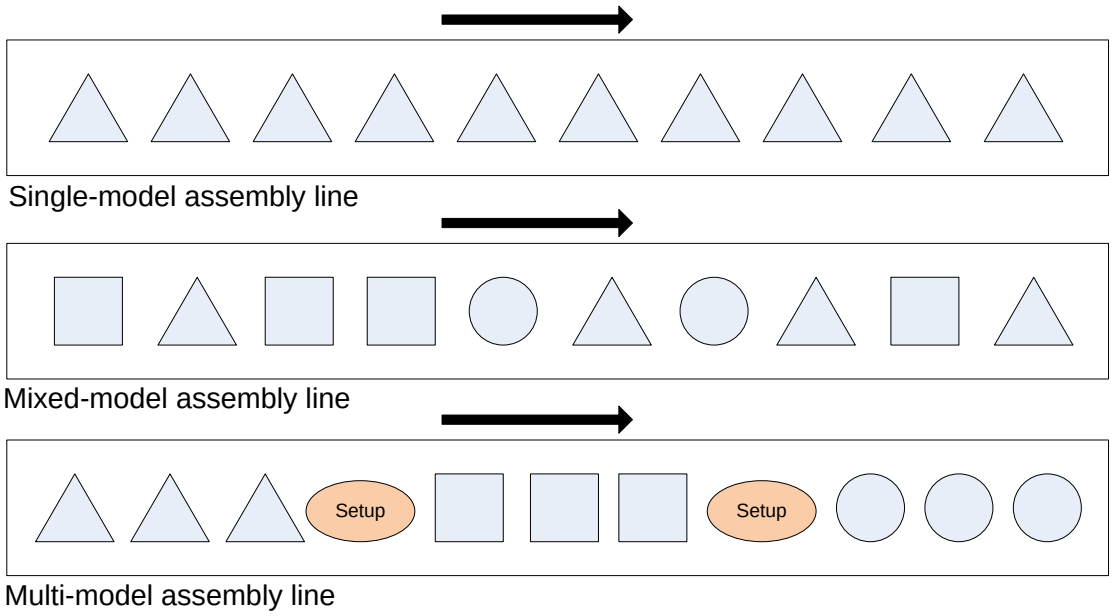


Figure 2.6: Assembly line for single, mixed and multi-model products (Becker and Scholl, 2006)

In mixed-model assembly line balancing (MMALB), most researchers proposed to solve the problem by combining the individual precedence diagram which represents different models into a single precedence diagram, called a joint precedence diagram (Battini et al., 2007). This approach can be found in most MMALB works (Haq et al., 2006; Su and Lu, 2007; Yagmahan, 2011).

Another approach to solving the mixed-model problem is using the "adjusted task time" approach. In this approach, the average task time for all models is used instead of actual task time for each model (Fokkert and Kok,1997). However, this approach is limited to the products that share similar precedence diagrams for all models, which rarely occurs in the real-world.

Researchers had employed different algorithms to optimise the MMALB problem. For instance, the Evolutionary Algorithm technique, mainly Genetic Algorithm, has been used in many previous works (Kim et al., 2000; Simaria and Vilarinho, 2004; Cao and Ma, 2008). Besides the well-known algorithm, GA is selected because of its ability to optimise complex problems (Venkatesh and

Dabade, 2008). However, the researcher also highlighted the drawback of GA such as high computational time in optimisation (Akgündüz and Tunali, 2011).

Researchers have also used Particle Swarm Algorithm (PSO) to optimise the MMALB problem (Sun, 2010; Chutima and Chimklai, 2012). The PSO is selected because of its simplicity and low computational time (Sun, 2010). However this algorithm cannot be directly used since the original algorithm is designed for continuous problems (Petropoulos and Nearchou, 2011). Other than that, researchers also applied Ant Colony Algorithm for this problem (Yagmahan, 2011). The simple heuristic approach had also been used for MMALB based on priority rules (Jonnalagedda and Dabade, 2010).

Based on previous optimisation works of MMALB, the ACO and PSO offer alternative algorithms to GA which are already used in many works. However, further testing is needed to identify the performance of both algorithms in MMALB, since there are only a limited number of existing works on it.

2.7 Research Trends

This chapter studied research in ASP and ALB which used soft computing approaches over the past 13 years. From this study, the previous research patterns and trends were identified. Figure 2.7 shows the number of published ASP and ALB papers which used soft computing methods between 2000 until the middle of 2013. The number of published papers in ALB shows a significant increase from 2006. Meanwhile, the ASP research papers show an incremental increase from 2009. This figure shows that, although the research in ASP and ALB were started early, it had been given special attention by researchers between 4-7 years ago. This trend is predicted to be maintained in the near future due to growth in computational techniques.

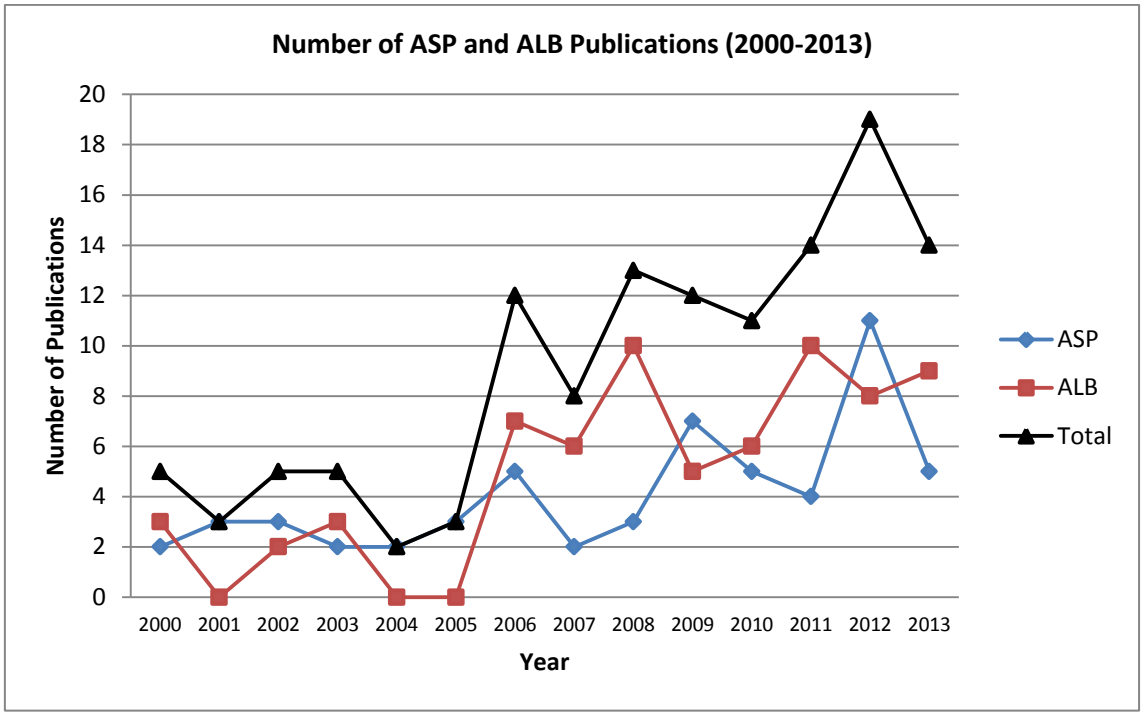


Figure 2.7: Number of published papers in ASP and ALB for 2000-2013

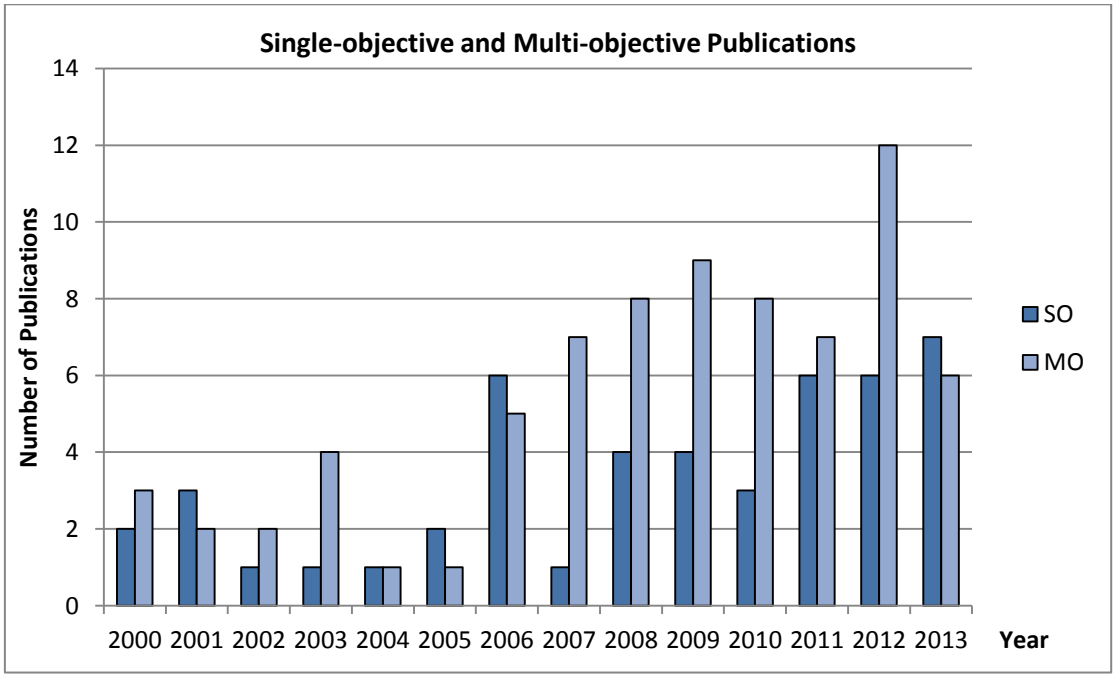


Figure 2.8: Number of papers that use Single and Multi-objective

Meanwhile, Figure 2.8 presents the trend of single-objective and multi-objective usage in ASP and ALB optimisation for 2000 until the middle of 2013. The trend

shows the number of research papers that used single-objective fluctuate from 2000 to 2008 and were stable afterwards. Meanwhile, a similar trend was also found in the number of papers that used multi-objective optimisation for the first five years. However, this trend changed during the second half of this period. The number of papers that used multi-objective optimisation started to grow from 2006. The multi-objective optimisation attracted many researchers because of the complexity of the problem and it is closer to the real assembly application.

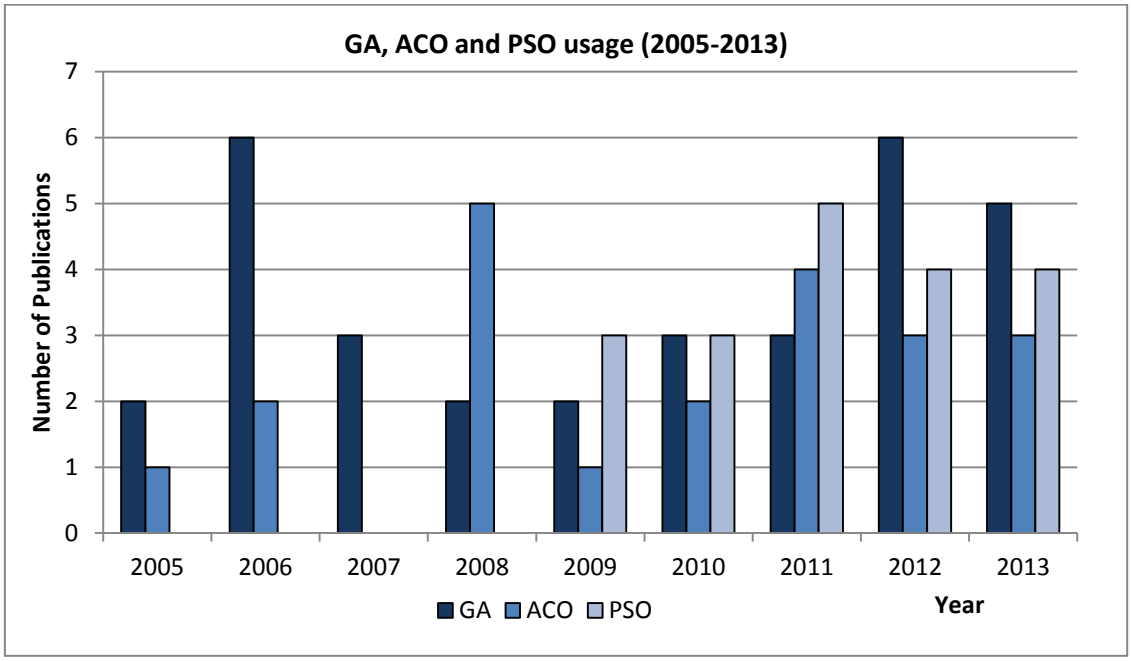


Figure 2.9: Number of papers that use GA, ACO and PSO for 2005-2013

In terms of optimisation algorithms usage, the application of GA in ASP and ALB papers between 2005 and 2013 is quite stable, with an average of three to four papers per year (Figure 2.9). For the same period, the ACO usage in the cited research papers fluctuates. Meanwhile, the PSO algorithm was first implemented in ASP and ALB research in 2009. The number of papers that applied PSO algorithm had shown rapid progress with three papers in 2009 and 2010, with five papers in 2011. In 2011, the papers using PSO in ASP and ALB optimisation outnumbered papers that employed GA and ACO. However, in 2012 the number of publications that used PSO was reduced compared to GA.

By the end of 2013 this number is predicted to increase as by the middle of 2013, the number of published papers using PSO was equal to the PSO usage for 2012.

A number of issues have also been raised by researchers regarding the ASP and ALB optimisation. One of the issues relates to the high computational time for ASP and ALB. Researchers agreed that the existing algorithms are maybe inadequate to solve larger ASP and ALB problems due to computational limitations (Chen and Liu, 2001; Capacho and Pastor, 2006; Tseng et al., 2010a; Toksari et al., 2010).

The second issue highlighted by researchers concerns tedious data entry procedure into computer programs. The current approach requires the researchers to identify and key in a set of data such as precedence, geometrical character, direction, etc. This process consumes a lot of time, as stated by a researcher; '*The man-computer interaction for constraint detection is the most manpower consuming process.*' (Su, 2009). To simplify the process, research on data extraction from the Computer-Aided Design (CAD) model is highly recommended by researchers (Wang and Liu, 2010; Chang et al., 2009; Su, 2009).

In addition, researchers also made an argument regarding assumptions in ALB. The first assumption stated that all workstations have similar capability. Therefore any assembly task can be assigned to any workstation. The idea of all workstations having similar facilities cannot be accepted because it does not reflect the real situation (Tseng and Tang, 2006). Meanwhile, Nearchou (2008) disagreed regarding the assumption that most of ALB problems are discussed as deterministic problems; in reality, the processing times are rarely deterministic.

Researchers in assembly optimisation have contributed to various problems and applications. However, there are still a few gaps and unfulfilled potential. ALB research started with simple line balancing problem with basic precedence constraint. This field has progressed to complex problems with other assembly

constraints. In computational experiment research such as ALB, the computational model is nearer to the actual situation when less assumption is used. However, the problem will become more complicated and requires higher computational cost. The suggestion of facilitating particular assembly tasks into particular workstations with facilities constraint has been discussed in earlier research, but it has not yet been implemented.

In ASP and ALB problems, optimisation algorithm plays an important role since both problems are classified as NP-hard. Research on algorithm improvement is important to handle more complicated ASP and ALB problems with larger size, various constraints and objectives. Currently, the algorithms used to optimise ASP and ALB problem are dominated by GA, ACO and PSO. The researchers are more interested in exploring and improving these algorithms although many other potential algorithms are available. However, the algorithm improvement works mainly focus on solving premature convergence issues rather than high computational time or algorithm complexity issues.

In the next few years, the algorithm usage is predicted to continue to be led by main algorithms (i.e. GA, ACO and PSO) with modifications to reduce premature convergence. Although there are many recent papers that focus on solving this problem, the definitive answer is still unclear. In the near future, the trend for algorithm hybridisation is also predicted to be the focus of research. The current success of hybrid algorithm has motivated researchers to give additional attention to this approach (Hui et al., 2009; Liu et al., 2009; Li and Shan, 2008). Although the algorithm hybridisation approach was started earlier, the number of papers using this approach has significantly increased since 2008. Therefore, plenty of opportunities exist in the algorithm hybridisation approach.

On the other hand, further research works on automation and integration of assembly optimisation also have the potential to be explored. At the moment, research on data extraction from CAD model are only being implemented at DFA level, but not widely used in ASP and ALB (Wang and Liu, 2010; Chang et al., 2009). Meanwhile, integration of assembly optimisation consumes larger

manpower to enter the data. At the same time, integration of assembly optimisation is also considered as a bridge to enable flows of extracted data from DFA level to ASP and ALB optimisation. Therefore, automation and integration of assembly optimisation are mutually dependent.

In summary, the research potential in ASP and ALB can be classified into two categories; (i) efficiency improvement and (ii) diversity of problem. The efficiency improvement works include algorithm enhancement and automation effort to reduce preparation or setup time. The problem diversity works comprise different problem formulations, problem complexity and problem integration.

2.7.1 Integrated ASP and ALB

One of the potential areas in ASP and ALB research is to diversify the problem by integrating ASP and ALB optimisation activities. Various benefits of ASP and ALB integration, as discussed in Section 1.4, have been emphasized to explain their importance. However, only a small number of research works focused on this topic.

Chen et al. (2002) proposed a hybrid GA to optimise assembly sequence planning and line balancing. The objectives are to minimise cycle time, maximise workload smoothness, minimise tool changes, minimise the number of tools and minimise the total penalty of assembly relations. Although the paper does not clearly state the integration of ASP and ALB, this relation is recognised based on objectives above.

In another research work, Tseng and Tang (2006) studied ASP together with ALB on the connector basis. In this research, optimisation was conducted in three stages. First, each part is assigned to a specific connector type. Then the algorithm will generate the assembly planning based on connectors. Finally, the algorithm assigns the connectors to stations and selects proper types of stations. In the second and third stages, GA has been applied to generate connector-based assembly in sequential order and to search for the station

types suitable for the sequential order (Tseng and Tang, 2006). However, by using this approach, when the connector number becomes larger, some parameters affecting the performance of the proposed GA should be reset.

Besides that, Tseng et al. (2008) were integrating ASP and ALB and optimising this problem using their proposed Hybrid Evolutionary Multi-objective Algorithms (HEMOAs) based on GA. Three main objectives were highlighted in the research. The objectives were to minimise direction changes, minimise tool changes and to minimise the workload difference among workstations. On the basis of multi-objective optimisation, the Pareto optimal approach was adopted in this study.

Another integrated ASP and ALB optimisation work was published in 2012 (Wang et al., 2012). The integrated problem was formulated based on assembly connectors as used in Tseng and Tang (2006). In this work, they proposed Guided-modified weighted Pareto-based multi-objective genetic algorithm (G-WPMOGA) to optimise the integrated ASP and ALB problem with two, three and four objectives. The proposed algorithm shows better performance in the problem with two and four objectives, but not in the problem with three objectives.

Recent optimisation work of integrated ASP and ALB shows that the GA-based algorithms performed well in optimising the problems with low and medium difficulties. However, when dealing with high difficulty problems, especially the problems with a large number of tasks, the GA-based algorithms did not perform very well. In this problem category (i.e. high difficulty problems), the Ant Colony Optimisation (ACO) algorithm performs much better compared to GA-based algorithms. However, the ACO only performed well in high difficulty problems, but not in low and medium difficulties (Rashid et al., 2012a).

2.8 Research Gaps

This research work will focus on integration of ASP and ALB optimisation, which are classified under the diversity of problem category. Both ASP and ALB

problems are categorised as NP-hard problems. In this case, the solution space is excessively increased when the number of components is increased. Based on this fact, optimisation of integrated ASP and ALB is likely to be more complicated compared to individual optimisation. Therefore, algorithm selection is crucial to ensure it produces high quality solutions in reasonable computational time.

In order to test the optimisation algorithm for integrated ASP and ALB, the availability of the test problem is critical to ensure the developed algorithm is well-tested before the final conclusion regarding the algorithm performance can be made. However, the test problems of integrated ASP and ALB from the literature is very limited due to the small number of published research work which focuses on this specific subject. In a number of ALB works, researchers have developed a problem generator to generate ALB test problem at various difficulty levels (Bhattacharjee and Sahu, 1990; Otto et al., 2011). Nevertheless, the existing test problem generator is limited to generating ALB test problems. Moreover, none of the existing work has developed a test problem generator for ASP problems.

In a number of individual ASP and ALB research works, PSO was proved to produce better solutions compared to GA in reasonable computational time (Liu and Wen, 2013; Li et al., 2013a; Hamta et al., 2013; Xing and Wang, 2012). At the moment, application of PSO to optimise ASP and ALB is performed individually. Although there are a few works on ASP and ALB integration, until now, no existing works used PSO algorithms to optimise the integrated ASP and ALB problems. Furthermore, there is no work on integrated multi-objective ASP and ALB using PSO which applies the Pareto optimal approach.

Besides that, the current research on integrated ASP and ALB optimisation is limited to single-model problems. The mixed-model for ASP individually is inapplicable because in normal practice, the assembly line issue was not considered in the Production Planning stage (see Figure 1.1). However, when the ASP and ALB are optimised concurrently, the possibility for considering mixed-model for integrated ASP and ALB arises.

2.9 Chapter Summary

This chapter has reviewed the literature in ASP and ALB optimisation. It has introduced the research in ASP and ALB, including the constraints and objectives used in optimisation works. In summary, this chapter has achieved the following goals:

- An overview of research in ASP and ALB optimisation has been provided.
- The representation schemes used to represent ASP and ALB problems have been identified.
- The soft computing methods used to optimise ASP and ALB have been analysed.
- The research trend and potential in ASP and ALB have been discussed.
- The research gaps in ASP and ALB optimisation have been identified.

CHAPTER 3

RESEARCH AIM, OBJECTIVES AND METHODOLOGY

This chapter presents the research aim and objectives based on research gaps identified from the literature review. Following that, the research scope and methodology in conducting this research is clarified. This chapter aims to achieve the following goals:

- State the research aim.
- Outline the research objectives.
- Clarify the research scope.
- Explain the research methodology.

3.1 Research Aim

The aim of this research is to establish a methodology and algorithm for integrating Assembly Sequence Planning (ASP) and Assembly Line Balancing (ALB) optimisation using Particle Swarm Optimisation. This research optimises both single-model and mixed-model integrated ASP and ALB problems.

3.2 Research Objectives

In order to achieve the research aim, the specific research activities are distributed into five objectives to cope with different issues. The research objectives are:

i. To establish an integrated representation scheme for ASP and ALB.
ii. To develop a tuneable test problem generator to generate integrated ASP and ALB test problems with wide a range of difficulties.
iii. To develop a multi-objective algorithm to optimise integrated ASP and ALB using Particle Swarm Optimisation.
iv. To extend the developed algorithm to optimise integrated mixed-model ASP and ALB problems.
v. To validate the performance of the proposed optimisation algorithm through test problems from literature and real-world problems.

3.3 Research Scope

This thesis explores the optimisation of integrated ASP and ALB using Multi-Objective Discrete Particle Swarm Optimisation (MODPSO). The review of the literature for this research focuses on the optimisation works for both problems which employed soft computing techniques from the year 2000 onwards. In addition, the problem is also limited to a simple model assembly problem (i.e. single-model in straight assembly line) which has accumulated a large number of works in the literature. The mixed-model assembly problem is also considered in this research to generalise the integrated ASP and ALB problem type.

The rest of the research scopes are addressed according to the following issues:

Integrated representation scheme: This research focuses on the establishment of an integrated representation scheme for ASP and ALB built based on assembly task. It considers the deterministic assembly information,

such as assembly direction, tool and assembly time. The assembly direction information is limited to six major axes (i.e. $+x$, $-x$, $+y$, $-y$, $+z$ and $-z$).

Tuneable test problem generator: This research develops a tuneable test problem generator to generate sufficient integrated ASP and ALB test problems with tuneable difficulty levels. In this case, the problem difficulty is controlled via four tuneable inputs (i.e. Number of Tasks, Order Strength, Time Variability Ratio and Frequency Ratio). Testing on the generated test problems is conducted using the existing algorithms which are used for individual and also integrated ASP and ALB.

Multi-Objective Discrete PSO: This research proposes Multi-Objective Discrete PSO (MODPSO) algorithms to optimise integrated ASP and ALB problems. During this stage, the author only considers the single-model assembly problem with a straight assembly line, which is the simplest version of assembly line problem. The MODPSO algorithm is tested with test problems generated from the test problem generator.

Integrated mixed-model ASP and ALB: In this research, the MODPSO algorithm application is extended to the integrated mixed-model ASP and ALB. It formulates the integrated mixed-model ASP and ALB using a joint precedence diagram which transforms a mixed-model into a single-model as practised in a majority of mixed-model works. Comprehensive testing, using problems from test problem generator is conducted.

Validation: Besides comprehensive performance testing using test problems generated from the test problem generator, this research validates the MODPSO algorithm using selected artificial test problems from the literature. In addition, the MODPSO is also validated using real-world problems. Since there are a limited number of integrated ASP and ALB problems in existing works, some of the selected problem data is randomly generated.

3.4 Research Methodology

The research methodology implemented in this work is summarised in Figure 3.1. This figure also maps the corresponding thesis chapter for each step.

3.4.1 Problem Identification

This research is in assembly optimisation with the purpose of improving the assembly efficiency. The problem addressed by this research is identified from the benefits of integrated ASP and ALB and also the advantages of the PSO algorithm.

3.4.2 Literature Review

The literature review is conducted by examining the reviewed journal and conference papers, thesis/dissertation and book chapters. It begins with an overview of ASP and ALB in order to understand the nature, importance and similarity of both problems. Next, the problem formulation and optimisation method for ASP and ALB are reviewed to identify the representation and optimisation methods that are used in existing works.

During this stage, the research activity is divided into two main tasks, as follows.

i. **Literature survey**. An extensive literature survey is performed to identify the research trend and potential in ASP and ALB optimisation. It starts with identification of keywords in assembly optimisation such as ‘sequence planning’, ‘line balancing’ and ‘production planning’. Based on the keywords, an extensive search of databases is conducted. The databases used for this search included Scopus, Google Scholar and Scirus. Next, the papers are filtered to select only relevant papers by screening the paper abstract. Finally the selected papers are fully reviewed and analysed to gain a clearer picture in the research area.
ii. **Identify the research gaps**. The research gaps in ASP and ALB optimisation are identified from the analyses and discussions of the literature survey.

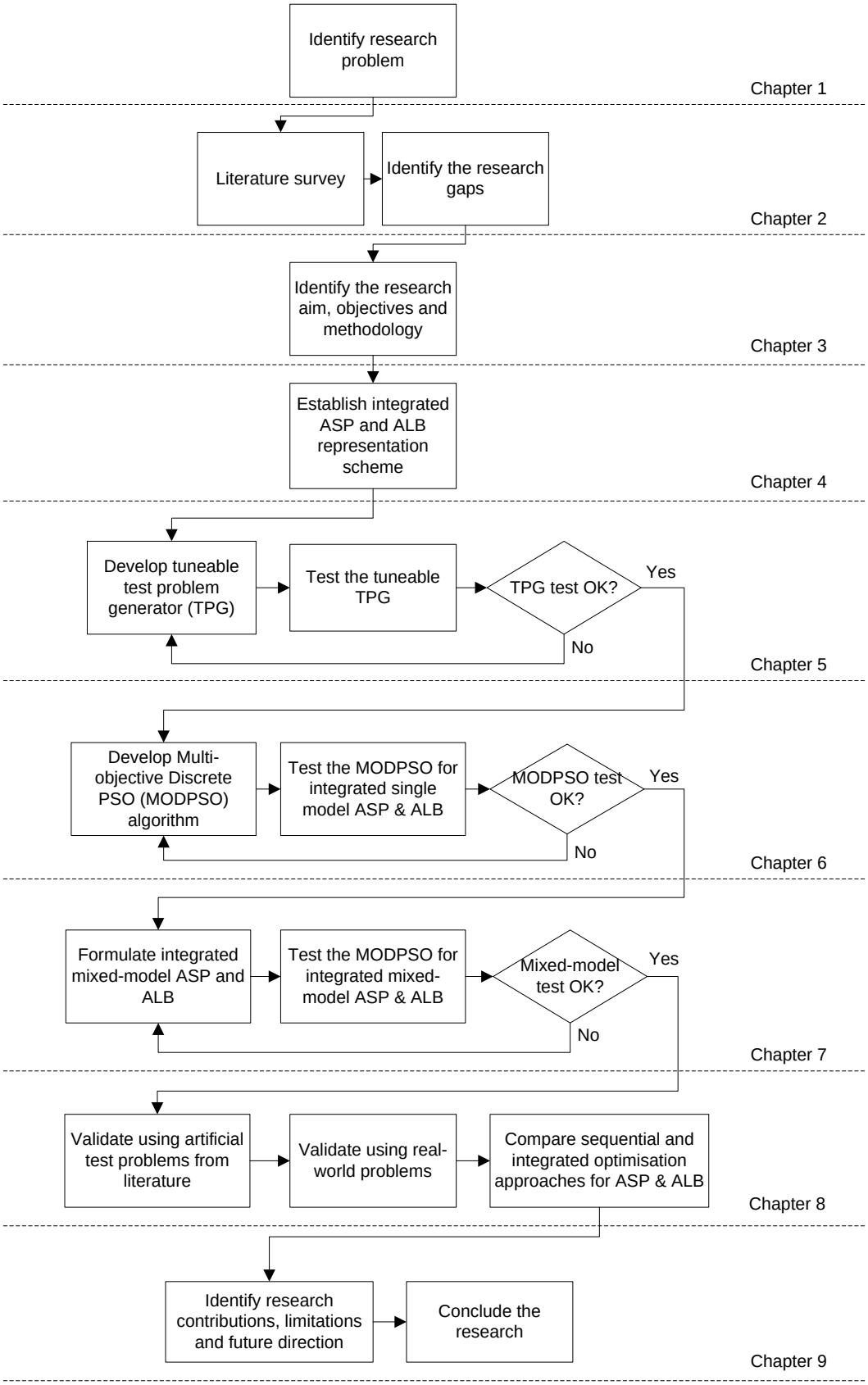


Figure 3.1: Research methodology and corresponding thesis chapter mapping

3.4.3 Identification of Aim, Objectives and Scope

The research aim and objectives are identified based on the research problem. In addition, the aim and objectives are clarified according to research gaps noted from the literature review. This ensures that the outlined research aim and objectives are in line with current research trends and tackle specific limitations of previous works. Since the research aim is stated in general terms, it is broken down into research objectives as a delivery strategy. The research scope explains the boundaries within which this research is constrained.

3.4.4 Establishment of Integrated ASP and ALB Problem Representation

In this research, the ASP and ALB problem representation is developed to fulfil the optimisation requirement according to the literature survey. The proposed representation scheme needs to represent ASP and ALB problems in order to enable integration of both problems. Besides that, the representation scheme must also be able to represent important attributes, as suggested in the literature survey. The application of the integrated representation scheme is demonstrated using an assembly example.

3.4.5 Development of Tuneable Test Problem Generator

The Tuneable Test Problem Generator (TPG) for integrated ASP and ALB is proposed to overcome the limitations of integrated ASP and ALB test problems from the literature. It will provide sufficient test problems with a wide range of difficulties for integrated ASP and ALB. The research in this stage is divided into two tasks.

i. **Develop tuneable TPG.** The details of TPG development methodology are explained, including its specifications, in Chapter 5. The proposed TPG must be able to generate integrated ASP and ALB problems at low, medium and high difficulty levels to ensure that the optimisation algorithm can be tested with different ranges of problem difficulty.

ii. **Test the tuneable TPG.** To ensure that the tuneable TPG is able to generate the integrated ASP and ALB test problem with different difficulties, a thorough test is conducted. This test inspects whether the tuneable input in TPG has an effect on the problem difficulty or not. In addition to that, the test also examines the ability of TPG to provide sufficient test problems in different difficulty ranges. The validity of TPG is very important to be tested at this stage because it directly influences the optimisation results in the forthcoming stages.

3.4.6 Development of Multi-Objective Discrete PSO Algorithm

In this stage, the research activity is divided into two main tasks, as follows.

i. **Develop Multi-Objective Discrete Particle Swarm Optimisation (MODPSO).** The algorithm to optimise ASP and ALB problems is developed in this stage. It explains the objective function, representation scheme, constraints and details of the proposed MODPSO algorithm. The Pareto-optimum approach is adopted in MODPSO to handle multi-objective problems.
ii. **Test the MODPSO for integrated single-model ASP and ALB.** In this research, the MODPSO algorithm is used to optimise multi-objective integrated ASP and ALB problems. To be more specific, a single-model integrated ASP and ALB that runs a homogeneous model on an assembly line is used. In this stage, the MODPSO algorithm is tested using the test problems generated from the tuneable Test Problem Generator. The performance of MODPSO is compared to six comparison algorithms. It is important to conduct the algorithm testing in this stage to ensure the performance of algorithms for the most basic problem type (i.e. integrated single-model ASP and ALB) before proceeding with more general types of problems.

3.4.7 Optimisation of Integrated Mixed-Model ASP and ALB

Once the performance of algorithms for integrated single-model problems is confirmed, the application of the MODPSO algorithm is extended to optimise integrated mixed-model ASP and ALB. The research activity during this stage is divided into two major tasks.

i. **Formulate integrated mixed-model ASP and ALB.** The integrated mixed-model ASP and ALB is formulated by transforming the individual precedence diagram into a joint precedence. The mixed-model assembly problem is selected because of the practicality of this problem to enhance product variety. In the real-world, an assembly line that only runs one model of product is rarely found, except for a model with an extremely high volume of demand.
ii. **Test the MODPSO for integrated mixed-model ASP and ALB.** For optimisation purposes, the MODPSO algorithm is altered to suit the integrated mixed-model ASP and ALB. The MODPSO algorithm is tested with the integrated mixed-model test problems and then compared to comparable algorithms.

3.4.8 Validation

Validation of the MODPSO algorithm is conducted to evaluate its performance using selected test problems. In this stage, three main research tasks are planned.

i. **Validate using artificial test problems.** For validation purposes, the author identified test problems used in the literature which cover a different range of difficulties. The test problems in the literature are classified into two categories; artificial test problems and real-world problems. In this research task, the validation is performed using artificial test problems from the literature.
ii. **Validate using real-world problems.** In addition to using artificial problems from the literature, the validation using real-world problems is

also conducted. However, because of the limited number of test problems that cover all the difficulty range, some of the assembly information is randomly generated. Also, some of the real-world problems are established by the author from real products.

iii. **Compare sequential and integrated optimisation approaches.** Finally, the validation is performed by comparing the integrated ASP and ALB optimisation approach with traditional sequential optimisation approach numerically. This is important in order to identify appropriate optimisation approaches for particular ASP and ALB problem categories.

3.4.9 Identification of Contributions, Limitations and Future Direction

Finally, the research contributions to knowledge are identified by reviewing the outcomes against the research aim and objectives. In addition, the research gaps are also examined to ensure how the contributions fill the stated gaps. Following that, the limitations of this research are identified, based on which the future direction for this research is discussed in order to overcome the limitations and enhance the research in this area.

3.5 Chapter Summary

This chapter highlights the research aim to establish methodology and algorithm for optimisation of integrated ASP and ALB problems using Multi-Objective Discrete PSO algorithm. The research aim is then distributed into smaller portions of action in research objectives and scope. Finally, the research methodology is presented as guidelines to be followed throughout this research.

CHAPTER 4

DEVELOPMENT OF INTEGRATED ASP AND ALB REPRESENTATION SCHEME

In any optimisation set-up, appropriate design of solution representation for the problem is crucial because it determines how well the optimisation process can go, determines the solution space and also the design space for the problem. In order to enable the ASP and ALB optimisation to be done concurrently, a single representation scheme that can be used to represent both of the problems is required.

The objective of this chapter is to establish an integrated representation scheme for ASP and ALB which incorporates essential optimisation information, so that the integrated optimisation can be performed. In order to achieve this objective, this chapter is written to fulfil the following goals:

- To propose the integrated ASP and ALB representation scheme.
- To explain the evaluation approach of assembly sequence based on the proposed representation.
- To demonstrate the application of integrated ASP and ALB representation to real life problems.
- To identify benefits and limitations of the proposed representation scheme.

4.1 Background

Although many works on assembly representation have been conducted, most of the papers present an independent representation scheme for ASP or ALB problems, built on the respective different basis. Most of the existing representation schemes for ASP are based on assembly parts, while ALB representation schemes are all based on assembly tasks, which lead to difficulties in integrating both problems. Several works that represent ASP problems are based on assembly tasks, but the optimisation objectives that are applicable to this type of representation are very limited. In order to enable the ASP and ALB optimisation to be done concurrently, a single representation scheme that can be used to represent both of the problems is required.

A survey of published works on ASP and ALB optimisation from 2000 until mid-2013 has been conducted and presented in Chapter 2. From this survey, the most frequent ASP optimisation objectives that have been used are to *minimise assembly direction change* and to *minimise number of tool change*. Meanwhile in ALB works, the dominant optimisation objectives are to *minimise cycle time*, *minimise number of workstations* and *minimise workload variance* (Rashid et al., 2012b). These findings show the importance and relevance of these objectives to be used in ASP and ALB optimisation. At the moment, none of the representation schemes offers to represent all of the attributes for these objectives except for Tseng and Tang (2006), who used connector-based representation. In connector based representation, the information presented is based on connector, rather than assembly part or task.

Other than that, Tseng et al.(2008) used task-based representation to represent assembly direction, tool and workload difference, but they do not clearly state how the assembly direction is defined for task-based representation, since they only used artificial problems (not representing real-world assembly). Therefore, it cannot be used to represent real-world problems without clear clarification as to how the assembly direction is determined for task-based representation.

This chapter proposed an approach to integrating ASP and ALB problems using a single representation scheme. This representation scheme will consider assembly direction, assembly tool, cycle time, number of workstations and workload variation as evaluated attributes. The proposed approach will be developed based on assembly task and combine the precedence graph and matrices. An integrated representation scheme for ASP and ALB enable both problems to be optimised together.

4.2 Proposed Representation Scheme

This section explains the proposed representation scheme for ASP and ALB. The proposed method will be used to transform an assembly product into measurable parameters according to optimisation objectives. The proposed method will result in an assembly representation that shows assembly tasks (design variables) in the form of a precedence graph (constraint) and also shows required data for ASP and ALB (optimisation parameters). The proposed method will also provide the evaluation strategy (objective functions) to evaluate the assembly sequence.

The proposed model is divided into two main sections; Representation and Evaluation. The representation scheme will be built based on the assembly task. To represent both ASP and ALB problems, the precedence graph will be used. Therefore, the main activity in the Representation section is to transform the assembly product into the task-based precedence graph as usually used in representing an ALB problem. The assembly data table will also be built to represent all the required data for ASP and ALB problems.

In order to build a task-based precedence graph from the assembly product drawing, any relation, connection or contact between one part and another needs to be determined. This information will be recorded in a liaison matrix, which is a modified version of Bourjault's liaison graph (Bourjault, 1984). Next, the precedence relation between liaisons is established using De Fazio's

question-and-answer procedure (De Fazio and Whitney, 1987). The rest of the proposed procedure is presented in Figure 4.1.

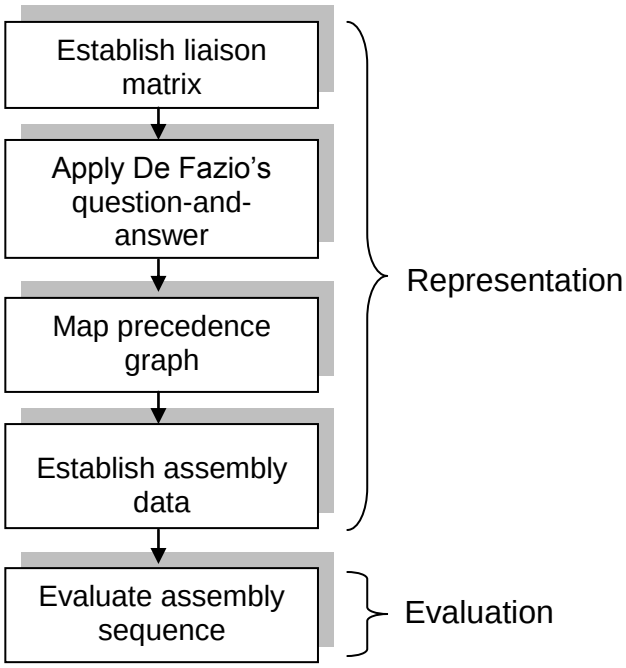


Figure 4.1: Flowchart of the proposed representation scheme

4.2.1 Basic Assumptions

Before presenting the proposed approach, certain assumptions need to be stated in order to define the representation scheme. The assumptions are listed as follows:

Assumption 1: For an assembly task, only two parts or subassemblies are involved. It might contain a combination of 'part and part', 'part and subassembly' or 'subassembly and subassembly'.

Assumption 2: Only one part or subassembly is moved during one assembly task. Therefore, one part or subassembly will be moving part and another one will be fixed part.

These assumptions are made to enable the assembly direction and tool to be used in task-based representation. Although the assembly direction and tool have been used in Tseng et al. (2008) in task-based representation, they only generated the data randomly for an artificial problem. Therefore, how the assembly direction is determined from a real product is not clearly defined.

4.2.2 Representation

All the ASP and ALB information such as design variables, constraint and optimisation parameters can be represented using precedence graph and data table. Therefore, the main activities in this stage are to establish a precedence graph and assembly data table from the product.

Establish liaison matrix

The idea of the liaison matrix is adopted from Bourjault's works in 1984 (Bourjault, 1984). However, in that work, Bourjault presented the liaisons in the graphical form known as a liaison graph. The liaison graph is one of the most successful approaches to representing ASP problems. Recent researchers who have presented liaison in the matrix format include Marian et al., (2006); Lai and Huang, (2004); Biswal et al., (2010); Lai and Huang, (2003).

However, previous works have only used Boolean expressions in the liaison matrix to determine the existence of assembly relationship. In the proposed method, the liaison matrix will be expressed using a unique numbering system. Besides showing assembly relations, the purpose of this approach is to give a unique number for each liaison that represents a particular assembly task. For a product with r parts, the relation between k^{th} and l^{th} parts is presented in the liaison matrix. If an assembly relation exists between k and l, $L(k, l)= a_i$ (i=1,2,...,n), otherwise, $L(k,l)$ is left blank. Here, n is the number of liaison (or assembly task) that exists in the assembly.

Apply De Fazio's question-and-answer

After establishing the liaison matrix, a question-and-answer (Q&A) procedure is applied to determine precedence relations in assembly tasks. This procedure is adopted from De Fazio and Whitney (1987), which consists of two questions for each task.

For task *i*;

Question 1: What tasks must be done prior to doing task *i*?

Question 2: What tasks must be left to be done after doing task *i*?

Map precedence graph

After answering these questions, all of the assembly precedence will be determined. For instance, in an assembly process with 5 tasks, the precedence constraint, C[(1,2),(1,3),(1,4),(2,3),(4,5)] shows that there are four precedence constraints to assemble a particular product. In this set, (1,2) brings information that task 1 must be done prior to task 2.

In a precedence graph, the assembly tasks are presented as a set of nodes with different labels inside the node which show the task number. Meanwhile, the precedence constraints are presented using the directed arcs. The outgoing arc from a particular node shows that the node is the predecessor for the node with incoming arc. For the above instance, the precedence graph nodes and

arcs are mapped as shown in

Figure 4.2. The predecessor task is represented with an outgoing arc and a successor task is shown by an incoming arc.

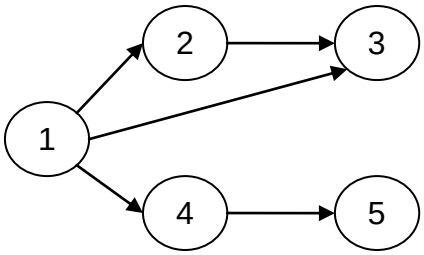


Figure 4.2: Precedence graph mapping

Next, the precedence graph needs to be updated by examining alternative routes from one node to another. According to the transitivity of the precedence constraints, the shortest paths between two generic nodes are removed (Fouda et al., 2001). Since there are two generic routes from 1 to 3 (1-2-3 or 1-3), the shortest route is eliminated from the precedence graph (Figure 4.3). The shortest route is eliminated because it does not bring any meaning until task 2 is performed.

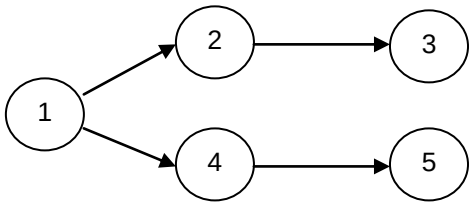


Figure 4.3: Updated precedence graph

Establish assembly data table

In this model, five optimisation objectives are considered. For ASP, the optimisation objectives are to (i) minimise the number of assembly direction changes and (ii) minimise number of tool changes. Meanwhile, for ALB, the objectives to (iii) minimise cycle time, (iv) number of workstations and (v) workload variation will be used for a given maximum allowable cycle time. In order to use these objectives, the required attributes are assembly direction (*D*), assembly tool (*T*) and assembly time (*M*).

As presented earlier, the assembly direction parameter is previously only used together with part-based representation in ASP. Since the proposed

representation is built based on assembly task, the assembly direction is redefined according to the basic assumption in Section 4.2.1. In contrast to previous works, the assembly direction is determined by assuming only one part or subassembly is moved during one assembly task. Therefore, the assembly direction based on assembly task can be defined.

The assembly data can be represented by a table sized n×3. In this table, n is the number of assembly task. The first column, D represents assembly direction. In this case, six major directions ($+x,-x,+y,-y,+z,-z$) are considered. Meanwhile the second and third columns shows assembly tool, T and assembly time, M respectively.

	D	T	M
a_1	D_{a1}	T_{a1}	M_{a1}
a_2	D_{a2}	T_{a2}	M_{a2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_r	D_{ar}	T_{ar}	M_{ar}

4.2.3 Assembly Sequence Evaluation

The main purpose of the assembly representation scheme is to enable the assembly sequence to be evaluated for the optimisation process. In this work, five optimisation objectives are considered according to frequently used objectives identified in a previous survey.

(i) Minimise number of assembly direction change

(ii) Minimise number of assembly tool change

(iii) Minimise cycle time

(iv) Minimise number of workstations

(v) Minimise workload variation

Given a feasible assembly sequence, the number of assembly direction changes is counted when the next assembly task requires a different assembly direction from the present assembly task. In this case, a similar approach also

applies to the second objective in determining the number of assembly tool changes.

To evaluate the third and fourth objectives, the maximum allowable cycle time, ct_{max} is required. According to Whitney (2004), cycle time is the time interval at which product units must be finished in order to meet demand. Normally, ct_{max} is determined from the number of demand or required output in the assignment period. Once ct_{max} is determined, the assembly tasks can be assigned into workstations. The cycle time (ct) for a particular assembly sequence is the highest processing time among all workstations. Processing time (pt) refers to total assembly time in a particular workstation. Once the total processing time for the current workstation is larger than ct_{max}, the present assembly task will be assigned to the next workstation. Then, the workload variation (v) for the fifth objective is calculated using the following formula. Here, nws refers to the number of workstations in the assembly line.

$$v = \frac{\sum_{i=1}^{nws} (ct\text{-}pt_i)}{nws}$$

Eq. 4.1[1]

4.3 Example of Application

In this section, an assembly example is presented to explain how to apply the proposed representation scheme. The example used is the assembly of a wall rack which consists of eight components including four fasteners, as shown in Figure 4.4. In assembling this product, two assembly tools are required. The first tool is a flat screwdriver used to assemble parts P4 and P5 and also parts P7 and P5. Meanwhile the second tool is a Philips screwdriver to assemble parts P3 and P1 and part P8 and P1. From Figure 4.4, an assembly liaison matrix in Table 4.1 is established.

[1]Eq. 4.1 is repeated from Eq. 2.5

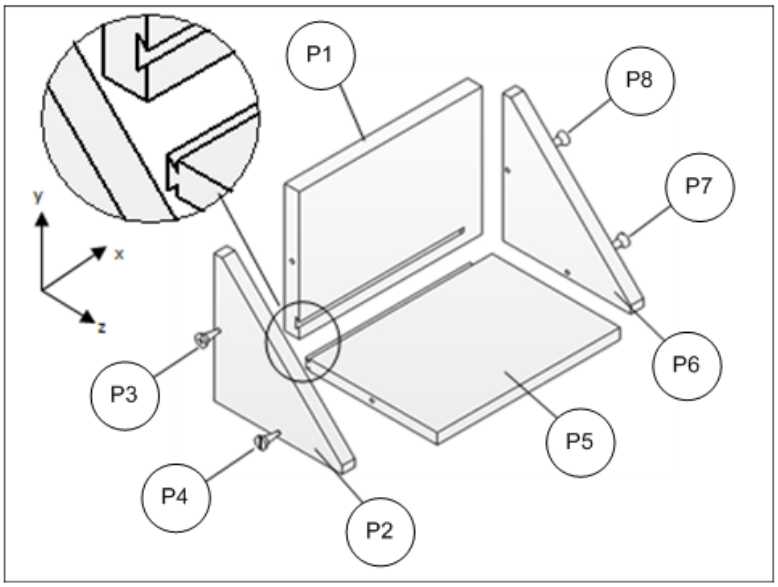


Figure 4.4: Assembly of wall rack

4.3.1 Assembly Problem Representation Example

***Establish liaison matrix*:** The liaison matrix is built by identifying whether there is any relationship between parts *k* and *l*. In this matrix, only the upper right side is filled since this is a reciprocal matrix, where $L(k,l) = L(l,k)$. For now, the assembly liaison is also known as assembly task.

Table 4.1: Liaison matrix for wall rack assembly

k \ *l*	P1	P2	P3	P4	P5	P6	P7	P8
P1	-	1	2		3	4		5
P2		-			6			
P3			-					
P4				-	7			
P5					-	8	9	
P6						-		
P7							-	
P8								-

***Apply De Fazio's question-and-answer procedure*:** For every task, the question and answer procedure is applied. For example in task 1, task 3 needs

to be done prior to task 1, while task 2 must be left until after task 1. The summary of De Fazio's question-and-answer (Q&A) is presented in Table 4.2.

Table 4.2: Summary of De Fazio's Q&A for wall rack assembly

Task	Answer for question:	
	Q1	Q2
1	3	2
2	1, 3	-
3	-	1
4	-	-
5	4	-
6	3	-
7	6	-
8	3	-
9	8	-

***Map precedence graph*:** From Table 4.2, the precedence constraints for this product can be identified. The precedence constraint for this problem is given as C[(3,1), (1,2), (3,2), (4,5), (3,6), (6,7), (3,8), (8,9)]. Then, the precedence graph is mapped.

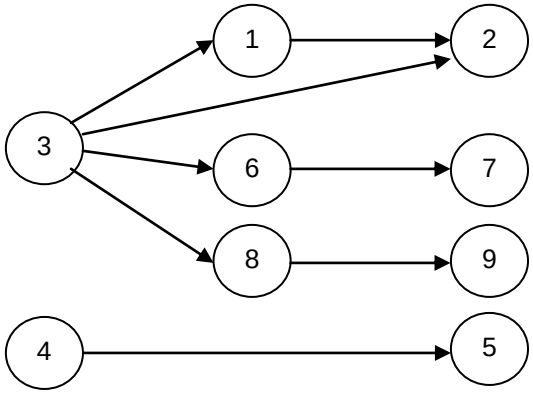


Figure 4.5: Precedence graph mapping for wall rack assembly

In Figure 4.5, one route can be classified as redundant, since there are generic routes to reach the specific nodes. The generic route is from 3 to 2. This node can also be achieved from 3-1-2. For this problem, the shortest route 3-2 is eliminated. The precedence graph for this problem after rearranging the nodes

is presented in Figure 4.6. Once the precedence graph is established, the design space that contains design variables and constraints is defined.

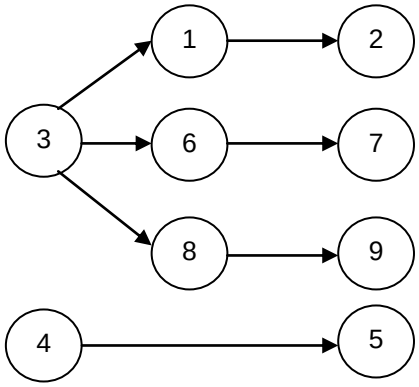


Figure 4.6: Precedence graph for wall rack assembly

***Establish assembly data table*:** The assembly data and information for this product is represented in Table 4.3. The assembly direction, *D* is obtained by determining the fixed and moving parts in each assembly task. For example, in determining assembly direction for assembly task 1, part P1 is defined as fix part and part P2 as moving part. Therefore, the assembly direction for task 1 is the direction of bringing part P2 to be assembled with part P1, which is in *+x* direction. Meanwhile, *T* is the assembly tool that is involved in the assembly task. For example, in performing assembly task 1, no assembly tool is involved, but to accomplish task 2, a flat screwdriver, labelled as *T1* is required. Then, the assembly time, *M,* is acquired by performing a time study of the product.

Table 4.3: Assembly data table for wall rack assembly

Task	*D*	*T*	*M (time unit)*
1	*+x*	-	4
2	*+x*	*T1*	12
3	*+x*	-	7
4	*-x*	-	4
5	*-x*	*T1*	12
6	*+x*	-	5
7	*+x*	*T2*	12
8	*-x*	-	5
9	*-x*	*T2*	12

4.3.2 Assembly Sequence Evaluation Example

To evaluate an assembly sequence, a feasible assembly sequence based on the precedence graph needs to be established. As an example, two feasible assembly sequences, F_1[3,6,4,5,1,2,8,7,9] and F_2[4,5, 3, 1, 6, 8, 9, 7, 2] are considered.

The first step in evaluating the optimisation objective is to assign the assembly tasks into workstations with the ct_{max} constraint. For each workstation, the total processing time (pt) must not exceed the ct_{max}. For given ct_{max}= 20 time unit, the example of assembly task assignment for F_1 is presented in Figure 4.7.

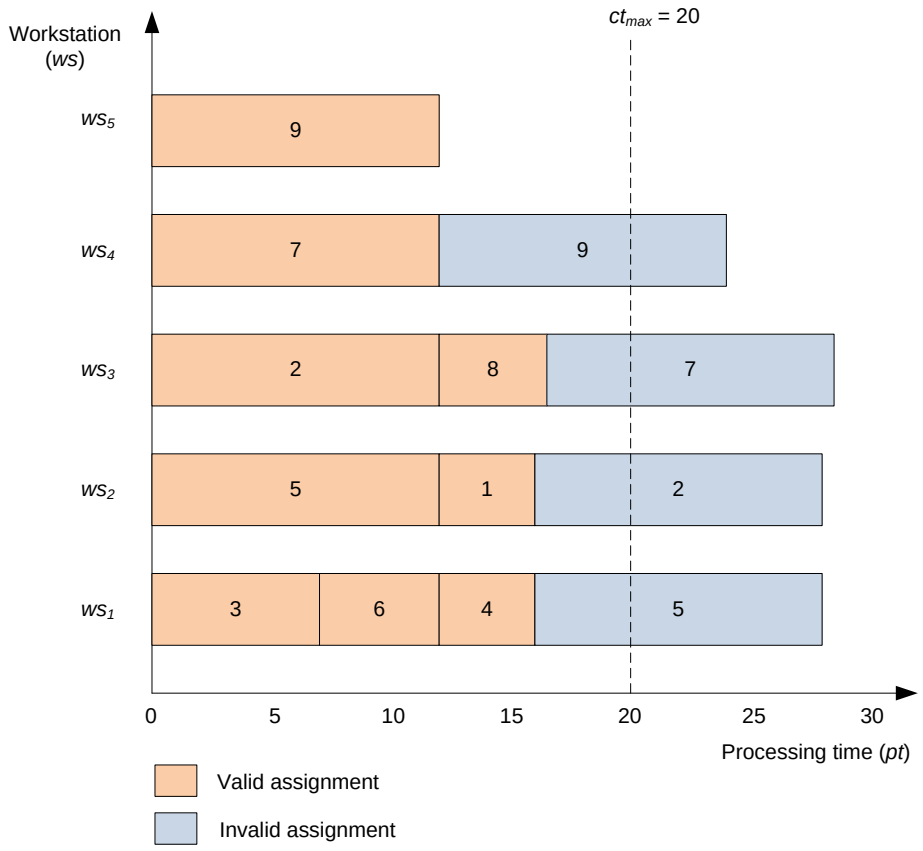


Figure 4.7: Assembly tasks assignment example for F_1

Based on Figure 4.7, for workstation 1 (ws_1) in F_1, the total assembly time for 3, 6 and 4 is 16 time units. If the assembly task 5 is also included in ws_1, the total assembly time will become 28 time units, which exceeds the ct_{max}. Therefore, the assembly task 5 is assigned into ws_2. Similar procedure is also applied to the subsequent workstations. The results of assembly tasks assignment for F1 and F2 are presented in Tables 4.4 and 4.5.

Table 4.4: Assembly task assignment for F_1

	ws_1			ws_2		ws_3		ws_4	ws_5
F_1	3	6	4	5	1	2	8	7	9
M	7	5	4	12	4	12	5	12	12
pt	16			16		17		12	12

Table 4.5: Assembly task assignment for F_2

	ws_1		ws_2			ws_3		ws_4	ws_5
F_2	4	5	3	1	6	8	9	7	2
M	4	12	7	4	5	5	12	12	12
pt	16		16			17		12	12

Number of assembly direction change (D_c) and tool change (T_c): The assembly direction change is calculated by summing up the number of assembly direction changes within all the workstations. The example of D_c and T_c calculation procedure is presented in Tables 4.6 and 4.7. For *nws* is total number of workstations and N_i is the number of tasks in workstation *i*:

$$D_C = \sum_{i=1}^{nws} \sum_{n=2}^{N_i} DC_n$$

Eq. 4.2

$$T_C = \sum_{i=1}^{nws} \sum_{n=2}^{N_i} TC_n$$

Eq. 4.3

For n^{th} assembly task in i^{th} workstation, the DC_n and TC_n

$$DC_n = \begin{cases} 0, \text{ if the direction of } n^{th} \text{ task is similar to the previous task} \\ 1, \text{ if the direction of } n^{th} \text{ task is not same as the previous task} \end{cases}$$

$$TC_n = \begin{cases} 0, \text{ if the assembly tool of } n^{th} \text{ task is similar to the previous task} \\ 1, \text{ if the assembly tool of } n^{th} \text{ task is not same as the previous task} \end{cases}$$

Table 4.6: D_c and T_c calculation example for F_1

	ws_1			ws_2		ws_3		ws_4	ws_5	
F_1	3	6	4	5	1	2	8	7	9	*Total*
D	*+x*	*+x*	*-x*	*-x*	*+x*	*+x*	*-x*	*+x*	*-x*	
Change	-	0	1	-	1	-	1	-	-	3
T	-	-	-	*T1*	-	*T1*	-	*T2*	*T2*	
Change	-	0	0	-	1	-	1	-	-	2

Table 4.7: D_c and T_c calculation example for F_2

	ws_1		ws_2			ws_3		ws_4	ws_5	
F_2	4	5	3	1	6	8	9	7	2	*Total*
D	*-x*	*-x*	*+x*	*+x*	*+x*	*-x*	*-x*	*+x*	*+x*	
Change	-	0	-	0	0	-	0	-	-	0
T	-	*T1*	-	-	-	-	*T2*	*T2*	*T1*	
Change	-	1	-	0	0	-	1	-	-	2

Assembly time: Based on assembly task assignment in Table 4.4 the highest processing time for F_1, *pt* in all workstations is 17 time units. Therefore, the cycle time, *ct* = 17 time units. Similar *ct* is also found in sequence F_2.

Number of workstation: To assign all the assembly tasks with ct_{max} constraint, five workstations are required in both sequences. Therefore the number of workstations for both F_1 and F_2 are equal to 5 workstations (*nws*= 5).

Workload variation: The workload variance, v_1 for sequence F_1 can be calculated using Eq. 4.1as follows:

$$v_1 = \frac{(17-16)+(17-16)+(17-17)+(17-12)+(17-12)}{5}$$

$$=2.4 \text{ time unit/workstation}$$

4.4 Discussion of Representation Scheme

In this chapter, an integrated representation and evaluation scheme for ASP and ALB is proposed. The proposed representation scheme is built on the basis of the assembly task, which is widely used in representing ALB problems. In this case, ASP representation which was usually based on assembly parts is transformed into assembly task. In transforming the ASP problem into task-based representation, the main problem is to determine the assembly direction because no previous work has represented assembly direction based on assembly task for a real-world product. This problem has been solved by redefining assembly direction according to Assumptions 1 and 2 (Section 4.2.1).

Previously, researchers have integrated ASP and ALB representations using a single representation scheme. They are Chen et al. (2002), Tseng et al. (2006), Tseng et al. (2008) and Wang et al. (2012). In contrast to Chen et al.(2002) who considered only assembly tools for ASP problem, the proposed representation also considered the assembly directions. This brings a new definition of this parameter associated with assembly task representation. Therefore, the proposed representation scheme provides more assembly parameters, which covers all important optimisation objectives.

In comparison with Tseng et al. (2008), they only represent artificial problems, where the assembly direction and tool data are randomly generated. Therefore, they do not clearly define how the assembly direction is determined in task-based representation.

Another integrated representation scheme proposed in Tseng and Tang (2006) and Wang et al.(2012) was based on assembly connectors. In this approach, the assembly parameters refer to the connectors. For example, the assembly direction and assembly time refer to connector direction and time to assemble the connector respectively. In contrast, the proposed representation considered the assembly parameters in terms of assembly task, which is closely linked to assembly process rather than connectors. This is because the connector-based representation requires all components to be grouped according to the

connector, but not all assembly components need a connector, as shown in the presented example. In this example, assembling parts P1 and P5 in Figure 4.4 operation does not utilise any connector.

In Section 4.3, an example has been presented in demonstrating how to transform an assembly product into the proposed representation. Besides that, the assembly evaluation is also shown according to predetermined objectives. This example shows that the ASP and ALB problem can be represented in a single representation scheme. In the proposed representation scheme, all information needed for ASP and ALB problems are represented using precedence graph and assembly data table. The design variable that is assembly sequences can be generated from the precedence graph. Meanwhile, the optimisation constraint which is precedence constraint is presented by a directed arc in the precedence graph. Then, the optimisation parameters are collected in the data table. In this case, the optimisation parameters from ASP problems are the assembly direction and assembly tool, while the assembly time is from the ALB problem.

The main benefit of the proposed representation scheme is the ability to represent assembly problems with all important optimisation objectives as identified in the literature survey for the real-world assembly problem. This is important to ensure that all the main optimisation objectives as used in individual ASP and ALB optimisations are taken into account. Besides that, the proposed representation scheme also used the task-based precedence graph as used in most of the ALB works. This simplified approach will aid workers in this field who wish to reuse data from existing work to optimise their problem without re-starting the data collection process.

Although the integrated representation scheme for ASP and ALB has been successfully developed, there is drawback to the proposed approach. The proposed approach to generate a precedence graph might become complicated for a large assembly task. However, the precedence graph for large assembly tasks can still be established using this approach. In the future, to simplify the

process, it is highly recommended that the precedence graph be generated automatically using the input of assembly task and precedence constraint.

4.5 Chapter Summary

The best approach to ensure simultaneous optimisation of ASP and ALB problems is by employing an integrated representation of solutions. A successful integrated approach will be able to combine and retain the important characteristics of the problem such as optimisation objectives which are usually used in previous works. This work shows that the proposed representation scheme is able to integrate ASP and ALB representation and at the same time consider all important optimisation objectives as used in individual ASP and ALB optimisation.

A small but important contribution behind the proposed representation is a new description of assembly direction by defining fixed and moving parts in the assembly task. By using this approach, the assembly direction parameter can be associated with task-based representation.

Therefore, the assembly task-based representation for ASP and ALB by considering all important optimisation parameters has successfully been proposed. In conclusion, this chapter has achieved the following goals:

- An integrated ASP and ALB representation scheme by considering the important objectives based on literature survey has been proposed.
- The assembly sequence evaluation approach based on the proposed representation has been explained.
- The application example of integrated ASP and ALB representation on real life problem has been shown.
- The benefits and limitations of the proposed representation scheme have been discussed.

CHAPTER 5

DEVELOPMENT OF TUNEABLE TEST PROBLEM GENERATOR

This chapter presents the development of a tuneable test problem generator (TPG) for integrated ASP and ALB. The literature review reveals the lack of test problems that cover a wide range of problem difficulties, especially for integrated ASP and ALB. Although algorithm development is important, any new algorithm should ideally be tested with a wide range of problem types before making any conclusion regarding their usefulness (Rardin and Uzsoy, 2001). Most of ASP and ALB works focus on proposing and demonstrating algorithm performance on specific ASP and ALB problems. There is a lack of investigation into testing and validating the performance of algorithms on wider classes of problems.

This chapter aims to achieve the following goals:

- Explains the requirements and specifications for the proposed TPG.
- Explains the methodology of the TPG development and example of application.
- Describes the experimental design to test the proposed TPG for ASP and ALB.

- ❖ Discusses the experimental results of the tuneable test problem generator.

5.1 Test Problem Generator Requirements

A TPG will be useful to provide a wide range of ASP and ALB problems with differing characteristics and difficulties. In many cases, the problem difficulty is only determined by the size of the problem (Ponnambalam et al., 2000; Tseng et al., 2004). While this is correct in certain cases, it overlooks the influence of many other attributes of problem difficulty. Additionally, TPG will also be useful to identify which algorithm may be more suitable for given types of problems. This knowledge is very important to help users to choose the right algorithm, and also for researchers to identify opportunities for further improvement in a particular algorithm.

To provide the mentioned benefits, the TPG must satisfy the following requirements:

i. Representation. The problems are generated on the basis of assembly task and represented using precedence graph. This is the common way to represent task-based assembly problem in earlier works (Rashid et al., 2012b).

ii. Output. The TPG is expected to produce precedence graphs that represent task-based assembly problems. Besides that, the TPG must also be able to generate assembly data, which consists of assembly direction and tool for ASP and assembly time for ALB. These types of data are selected based on popularity from the literature survey (Rashid et al., 2012b).

iii. Tuneable difficulty level. One of the important features expected in a TPG is tuneable difficulty level. This feature will ensure that test problems are generated within known difficulty ranges as required.

Not many proposals exist in the literature regarding methods for generating test problems in this domain. Furthermore, existing proposals are limited to generating test problems for ALB. Bhattacharjee and Sahu's proposal is to generate a random precedence graph to represent an ALB problem (Bhattacharjee and Sahu, 1990). In this approach, the assembly problem is generated randomly and the problem difficulty measured to determine its complexity level. Later, Otto et al. (2011) proposed a systematic data generator for assembly line balancing. Besides presenting a systematic method for generating precedence graphs, this work also demonstrates that common graph structures in real-world assembly problems, i.e. chains, bottlenecks and modules can be generated on a precedence graph. This approach is also able to generate problems at the desired difficulty levels (Otto et al., 2011).

Otto's work is the one closest to our stated requirements because this work fulfils the stated requirements (i) and (iii). In Otto's work, ALB problems are generated based on assembly tasks and represented using precedence graphs. It also gives users the ability to create test problems difficulty at the desired level of difficulty. However, since this work was specifically developed for ALB problems, it fails requirement (ii). Therefore, the ALB-only systematic data generator proposed by Otto in 2011 will be expanded to incorporate both ASP and ALB test problems.

5.2 Test Problem Generator Development

The test problem generator is developed using the methodology presented in Figure 5.1. The details of each step are explained in Sections 5.2.1 to 5.2.5. The first step in developing the TPG is to identify the input and output elements. Next are the independent development of automated generators for assembly graph, ASP and ALB data. Finally, the outputs from graph and data generators are synchronised and combined to produce a complete test problem set.

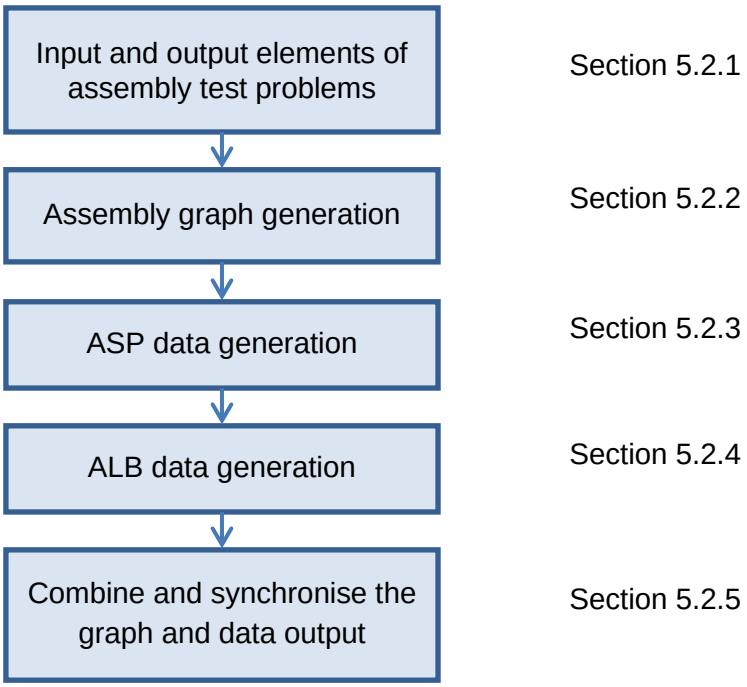


Figure 5.1: Test problem generator development flow

5.2.1 Input and Output Elements of Assembly Test Problems

The mapping of input and output variables is shown in Figure 5.2. The tuneable inputs are presented in bold and italic font.

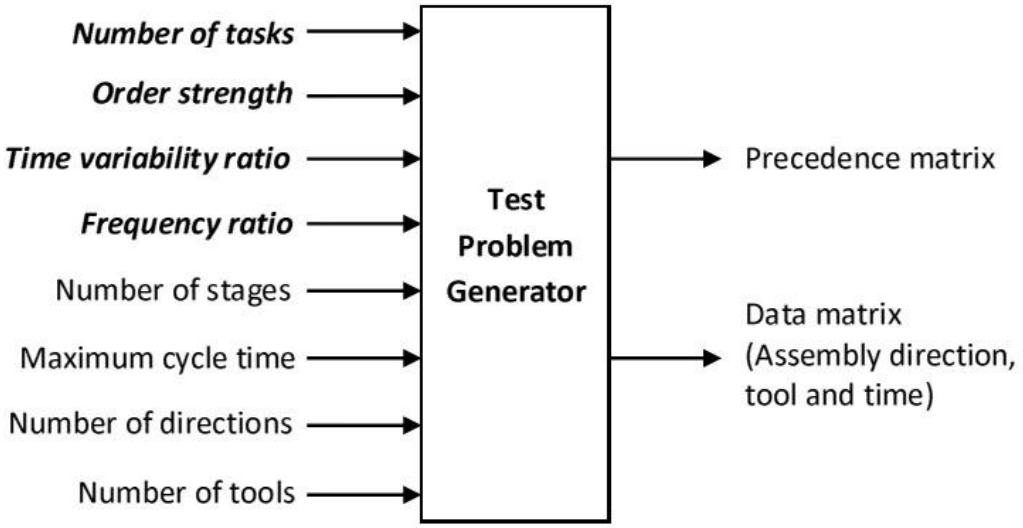


Figure 5.2: Test problem generator input and output map

Tuneable Input Elements

The tuneable input elements are variables used to control the problem difficulty generated by the TPG. In this work, one new tuning variable is proposed and the rest are adopted from previous works. The TPG is conceptually divided into two parts: the generation of assembly graphs and the generation of assembly data. The next section will discuss the tuneable input variables for each part. Although the tuneable input variables for ALB have been discussed in earlier works, no clear link has been suggested in the literature between input and specific difficulty levels for ASP (Scholl, 1993).

Tuneable Input for Assembly Graph

Two tuneable inputs will be used to generate precedence at a specific complexity level. The first input variable to measure graph complexity is *n*, the

number of nodes in a graph. In ASP and ALB contexts, graph nodes represent assembly tasks for a given problem. The number of possible assembly sequences will exponentially increase with the number of nodes. In the surveyed literature, the size of ASP problems varies between five and 75 nodes; in ALB, 86% of surveyed ALB papers used between seven and 150 nodes, while the remaining 14% used up to 300 nodes.

Another graph input variable conceptually linked to graph difficulty is Order Strength. Order Strength (*OS*) measures the relative number of precedence relations in a graph. By increasing the relative number of precedence relations, the resulting graph is expected to be more complicated (Scholl, 1993; Bhattacharjee and Sahu, 1990). *OS* is defined as a total number of ordering relation in transitive closure divided by the possible number of ordering relations for particular graph. The *OS* is calculated as follows.

$$OS = \frac{R}{P}$$

Eq. 5.1

R – Total number of ordering relations

P – Possible number of ordering relations

$$P = \frac{n(n\text{-}1)}{2}$$

Eq. 5.2

n – Number of nodes

The *OS* value varies between [0, 1]. *OS* = 0 shows that there is no precedence relation in the graph and *OS* = 1 shows that there is only one feasible sequence for the particular problem. The *OS* attribute is used together with *OS* tolerance (δ_{OS}) since it is difficult (impossible in some cases) to meet the exact *OS* value.

Tuneable Input for Assembly Data

Previously, a number of time-related measures for ALB data have been proposed, such as ratio between maximum and minimum completion time between assembly lines, standard deviation, and Time Variability ratio (Kilbridge and Wester, 1961; Bhattacharjee and Sahu, 1990). Time Variability ratio (*TV*) has consistently been used in previous works and is selected for use in this work. *TV* indicates the range of task time of all tasks dispersed between the assembly lines. *TV* is calculated as follows:

$$TV = \frac{t_{max}}{t_{min}} \qquad \text{Eq. 5.3}$$

$$t_{max} \geq \frac{1}{3} ct_{max} \qquad \text{Eq. 5.4}$$

t_{max} – maximum task time
t_{min} – minimum task time
ct_{max} – maximum cycle time

A smaller *TV* value indicates that existing task times are distributed in a smaller range, which leads to an increased level of problem complexity. The t_{max} constraint in Eq. 5.4 is introduced to avoid the generation of uniformly small task time, which leads to inconsistency of difficulty levels. The ct_{max} constraint is explained in the following section.

Meanwhile, in the ASP problem domain, no variable for measuring data complexity has been established. In this work, the ASP data considered are assembly directions and assembly tools. This type of data can be measured by considering how many times (i.e. frequency) a similar direction or tool appears in the problem. A common optimisation objective is to minimise direction or tool changes in a sequence of tasks. Thus, the Frequency Ratio (*FR*) is proposed to be used as an input variable that measures ASP data complexity.

$$FR = \frac{f_{min}}{f_{max}}$$

Eq. 5.5

f_{min} – Minimum data frequency

f_{max} – Maximum data frequency

Data with a higher *FR* value is harder to arrange to achieve a minimum number of changes because the choice and variability of data are high. This type of data will usually produce a higher number of changes compared to smaller *FR* data. The details of graph and assembly data attributes level are shown in Table 5.1. In this table, the attribute level for 'number of nodes' is proposed based on a survey on problem sizes as mentioned earlier in the *Tuneable Input for Assembly Graph* Section, while the proposed classification of *FR* and *TV* levels is based on a few initial tests. The proposed classification of *OS* levels is adopted from the literature review (Otto et al., 2011).

Table 5.1: Assembly graph and assembly data attribute levels

Attributes	**Low**	**Medium**	**High**
Number of nodes, *n*	$n \leq 20$	$20 < n \leq 70$	$n > 70$
Order strength, *OS*	$OS \leq 0.2$	$0.2 < OS \leq 0.6$	$OS > 0.6$
Time variability ratio, *TV*	$TV > 6.5$	$2.5 < TV \leq 6.5$	$TV \leq 2.5$
Frequency ratio, *FR*	$FR \leq 0.2$	$0.2 < FR \leq 0.6$	$FR > 0.6$

The tuneable input variables are loosely classified into Low, Medium and High levels because of the non-linearity of the problem. Although the general trend of problem difficulty over tuneable variables can be predicted, when tuning for a targeted difficulty level, too small variable changes may lead to inconsistent difficulty levels. The classification of level difficulties as in Table 5.1 can be used

as a guideline for users in selecting appropriate difficulty levels for their use. To reduce the possibility of inconsistent difficulty levels, it is suggested to use the mid-point of the Medium level to generate Medium difficulty problem.

Other Input Elements

Apart from the tuneable elements, other 'compulsory' inputs are required for generating a complete problem. Although some of these variables have implications for the problem difficulty level, they are not used here as a means to control the problem difficulty because of a lack of agreement in the literature. These inputs are: number of stages (s), maximum cycle time (ct_{max}), number of assembly direction (n_{dir}) and number of assembly tool (n_{tool}). Number of stages (s) refers to number of column that contains nodes in a specific precedence graph. In Figure 5.3, the example graph consists of three stages (hence s=3), shown separated by dotted lines. This variable determines the basic shape of graph, where a smaller number of stages will produce graphs with more parallel nodes. The *maximum cycle time* (ct_{max}) is the upper limit of allowable cycle time. This variable is calculated from the required production rate of the assembly line. The *number of directions* and *number of tools* are also required to generate ASP data.

Another important element of the TPG is the pseudo-random number generator that underlies most of the data generation algorithm. In this work, the pseudo-random generator used is the Mersenne Twister with a range between [0, 2^{32} - 1] for 32-bit integer (Panneton et al., 2006). Appropriate use of seed values ensures that all results are reproducible.

Output Elements

Two sets of outputs are generated by the proposed TPG. The first output is the assembly precedence graph (e.g. Figure 5.3), represented by a precedence matrix, which is an $n \times n$ matrix filled with '1' or '0' values (Table 5.2). The leftmost column shows assembly tasks and the top row shows the follower tasks. The value '1' shows that task j must be performed after task i.

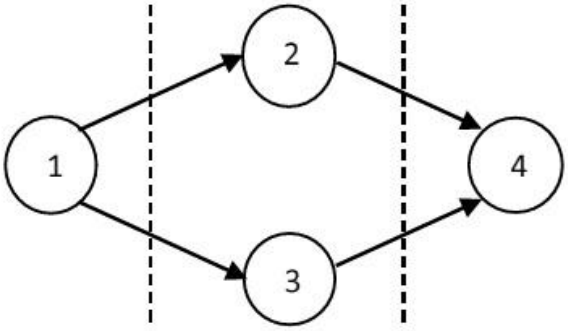


Figure 5.3: Example of precedence graph

Table 5.2: Example of precedence matrix

i	*j*			
	1	**2**	**3**	**4**
1	0	1	1	0
2	0	0	0	1
3	0	0	0	1
4	0	0	0	0

The second output is a data table that consists of assembly directions, assembly tools and assembly time associated with every task. This data is generated according to the required difficulty level as determined in tuneable input variables.

5.2.2 Assembly Graph Generation

In this work, the systematic graph generation method is adopted from Otto's work (Otto et al., 2011), from which the five steps below are proposed.

Step 1: Provide all the compulsory inputs. The compulsory inputs are number of nodes (n), desired Order Strength (OS_d), Order Strength tolerance (δ_{OS}) and number of stages (s).

Step 2: Generate and distribute the nodes in all stages using uniform distribution.

Step 3: Connect every node in stage k>1 with exactly one random node in stage k-1. This step is important to keep the nodes in their original stages.

Step 4: Calculate the *OS* using Eq. 5.1. If the *OS* is within $OS_d \pm \delta_{OS}$, then terminate the process. Otherwise, continue with Step 5.

Step 5: Select a node i in stage $k < s$ and insert an arc to a random node j in stage $m>k$ until the desired *OS* is achieved. A direct arc from node i to node j is allowed only if:

1. Task i has no restriction such as isolated node or special structure.

2. The *OS* values have not exceeded the desired upper limit.

5.2.3 ASP Data Generation

In this work, the ASP data considered are 'assembly direction' and 'assembly tool change'. The following steps are applied to generate this data. Besides number of tasks, n, the required input in ASP data generation is 'Frequency Ratio'*, FR*.

Step 1: Calculate all possible lower (L_{limit}) and upper (U_{limit}) limits of data frequencies according to *FR*. The L_{limit} and U_{limit} represent the minimum and maximum number of times that a particular direction or tool appear in the generated problem. These limits must fulfil the following constraints:

$$U_{limit} \times (n_{type} - 1) + L_{limit} \geq n \quad \text{Eq. 5.6}$$

$$L_{limit} \times (n_{type} - 1) + U_{limit} \leq n \quad \text{Eq. 5.7}$$

Eq. 5.6 and 5.7 ensure that the summation of generated data within upper and lower limits matches the number of tasks, n. In these equations, n_{type} represent the number of direction (n_{dir}) or number of tool (n_{tool}) type. In this work, six major

direction axes (+*x*,-*x*,+*y*,-*y*,+*z*,-*z*) are considered, thus n_{type} for n_{dir} is equal to six. Meanwhile the n_{type} for n_{tool} depends on the number of tool types in a particular assembly line.

Step 2: Randomly select a pair of lower and upper limits from the set of possible limits determined in Step 1. Generate remaining data frequencies using uniform distribution. The summation of data frequencies must be equal to *n*.

Step 3: Generate the ASP data based on frequencies (Step 2) in random order.

5.2.4 ALB Data Generation

The ALB data to be generated is the 'task time' for all nodes. The required inputs are 'maximum cycle time' (ct_{max}) and 'Time Variability ratio' (*TV*). This data is generated in two steps:

Step 1: Calculate all possible limits of task time based on *TV*. The upper limit must not exceed ct_{max}. Randomly select an upper and lower limit from all possible limit pairs.

Step 2: Generate the remaining task times between upper and lower limit using uniform distribution.

5.2.5 Combine and Synchronise the Graph and Data Output

Synchronisation of ASP-specific and ALB-specific outputs is straightforward because ASP and ALB representations are both developed using the same assembly task basis (Rashid et al., 2011). Data generated in Sections 5.2.3 and 5.2.4 is directly linked with assembly tasks and no further adjustment is needed. In this synchronisation step, the output data consisting of ASP data and ALB data are combined to establish a data table. In the data table, the assembly direction data is located in the first column, assembly tool data in the second column and assembly data for ALB in the third column.

The final process in this step is to transform the precedence graph into a precedence matrix as explained in the ***Output Elements*** section. This is an important process designed to synchronise the format of assembly graph into a readable computer language.

5.3 Test Problem Generation Example

This section demonstrates how the test problem generator works using a selected example with 15 nodes.

5.3.1 Input and Output Elements

The precedence graph that represents the assembly problem is set at medium *OS* level. In this example, the compulsory attributes is set as follows:

n =15	OS_d = Medium
OS tolerance (δ_{OS}) = 0.05	Number of stage, s = 3
FR_{dir} = Medium	FR_{tool} = Medium
ct_{max} = 55 time unit	TV = Low

5.3.2 Assembly Graph Generation

Once the number of tasks and number of stages are defined, the graph nodes are randomly distributed among all stages based on uniform distribution. In this example, the generated nodes distribution are *nd*= [3 7 5]. Each element in *nd* shows the number of nodes in specific stage.

In the next step, each of the nodes in stage k> 1 is randomly connected with tasks in stage k-1. For example, node 7 in stage 2 is connected with node 2 in stage 1. This process is repeated for all nodes in stage > 1. The completed generated precedence graph is presented in Figure 5.4.

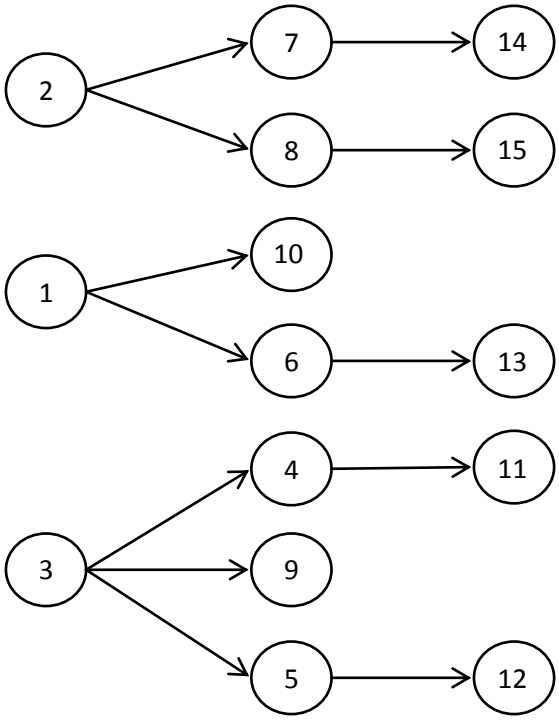


Figure 5.4: Initial generated precedence graph at low *OS*

Next, the Order Strength for the graph is calculated using Eq. 5.1. The total number of ordering relation (*R*) for this example is 17. The ordering relation for a particular node is the total of direct and indirect followers for that node. For node 1, there are three nodes that have ordering relation, node 6, node 10 and node 13. The total ordering relation for node 2 is four. The total number of ordering relations for this graph is calculated by summing up the ordering relation for each node. While the possible number of ordering relations according to Eq. 5.1 is 105. Therefore, the *OS* for this example is 0.162.

Based on the calculated *OS*, the generated graph is still classified in low difficulty because the $OS \leq 0.2$. To increase the *OS* value, a node from stage $k<s$ is randomly selected and then connected with a random node from higher stage. This procedure is repeated until the desired *OS* level is achieved. In this example, the precedence graph with medium *OS* level is generated, as in Figure 5.5. The *OS* value for this graph is 0.275.

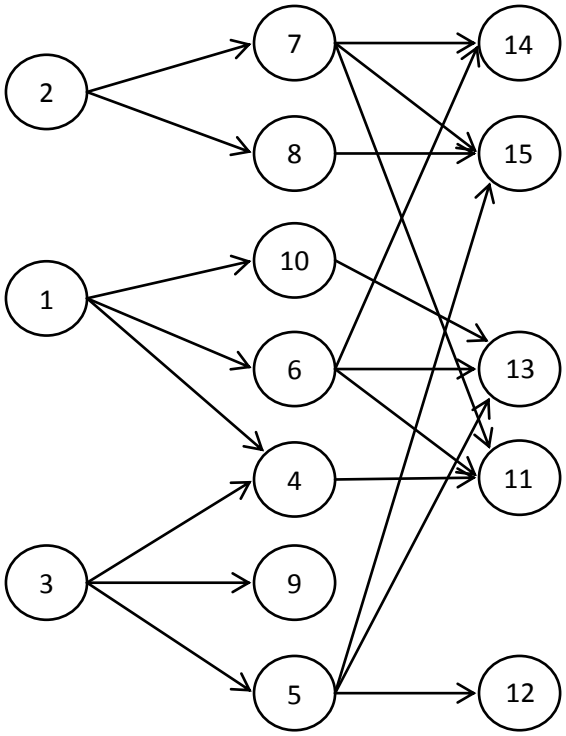


Figure 5.5: Generated precedence graph at medium *OS*

5.3.3 ASP Data Generation

The ASP data considered in this work are assembly direction and assembly tool change. For assembly direction, the six major directions taken into account are *+x,-x,+y,-y,+z* and *–z*. At the same time, six tool types are used. In this example, the assembly direction and tool is set at medium level using *FR*. To generate the data, the first step is to generate lower and upper limit for data frequency by fulfilling the constraint in Eq. 5.6 and 5.7. Based on *FR* value, the possible upper and lower limit for direction and tool data are [(1,3)(1,4)(1,5)(2,4) (2,5)]. Then a set of data limit is randomly selected from the possible set above. In this example, the selected frequency limit for assembly direction data is (1,4) and for assembly tool data (2,5). Next, the remaining data is generated between lower and upper limits. The total frequencies for both data must be equal to the number of nodes. The generated assembly direction frequencies is [4 3 1 3 3 1] and assembly tool frequencies is [2 5 2 2 2 2]. Finally, the data is randomly distributed among all nodes based on the above frequencies.

5.3.4 ALB Data Generation

Meanwhile, in generating assembly time data for ALB problems, the required inputs are maximum cycle time (ct_{max}) and Time Variability ratio (*TV*). In this example, the ct_{max} is set at 55 time units and *TV* is at low level (*TV*>6.5). In order to find the lower and upper limits for assembly time data, two tasks time are randomly generated between [1, ct_{max}]. Then the *TV* is calculated using Eq. 5.3. This step is repeated until the desired *TV* ratio is achieved. In this example, the lower and upper limit is (5, 36). Next, the remaining data between lower and upper limit is generated using uniform distribution. The generated ASP and ALB data for this example is shown in Table 5.3.

Table 5.3: Generated ASP and ALB data

Node	Direction	Tool	Time
1	*-x*	*T5*	22
2	*+x*	*T1*	8
3	*-x*	*T2*	17
4	*-y*	*T2*	10
5	*+z*	*T3*	11
6	*-x*	*T2*	15
7	*+x*	*T3*	31
8	*+z*	*T2*	8
9	*-y*	*T6*	9
10	*+z*	*T6*	5
11	*-y*	*T1*	20
12	*+x*	*T2*	16
13	*-z*	*T4*	36
14	*+y*	*T5*	16
15	*+x*	*T4*	10

5.4 Experimental Design for TPG

This section describes the set-up of the experimental design to assess problems generated using the proposed test problem generator (TPG). The experiment is divided into two phases. In Phase 1, the experiment will focus on the ability of TPG to generate problems at the desired complexity level by manipulating the tuneable input attributes. Then, in Phase 2, the generated problems from TPG will be used to evaluate the performance of a set of selected algorithms. The purpose of the second phase experiment is to identify whether the generated problems from TPG can be used to characterise the best and worst performance of each algorithm.

5.4.1 Phase 1: Testing of Tuneable Input

The experiment in this phase is conducted by dividing all the tuneable input variables into five levels, as presented in Table 5.4.

Table 5.4: Tuneable input level setting

Level	*n*	*OS*	*TV*	*FR*
1	15	0.2	2	0.2
2	20	0.3	3	0.3
3	40	0.4	4	0.4
4	60	0.5	6	0.6
5	80	0.6	8	0.8

A reference variable setting (datum) is selected as a baseline, while the rest of the problem variable settings are generated by changing only one variable value at a time. In this case, level 3 is selected as the reference variable setting because it sits at the middle between minimum and maximum values. The complete experimental table for Phase 1 is shown in Table 5.5.

Table 5.5: Experimental table for Phase 1

Problem	*n*	*OS*	*TV*	*FR*
1	**40**	**0.4**	**4**	**0.4**
2	15	0.4	4	0.4
3	20	0.4	4	0.4
4	60	0.4	4	0.4
5	80	0.4	4	0.4
6	40	0.2	4	0.4
7	40	0.3	4	0.4
8	40	0.5	4	0.4
9	40	0.6	4	0.4
10	40	0.4	2	0.4
11	40	0.4	3	0.4
12	40	0.4	6	0.4
13	40	0.4	8	0.4
14	40	0.4	4	0.2
15	40	0.4	4	0.3
16	40	0.4	4	0.6
17	40	0.4	4	0.8

From Table 5.5, 17 test problems are generated by changing one variable at a time. Problem 1 represents the reference variable setting, problems 2 – 5 examine the effect of *n*, problems 6 – 9 for effect of *OS*, problems 10 – 13 for effect of *TV* and problems 14 – 17 for effect of *FR*.

In order to solve precedence graphs, the topological sort algorithm is used to generate feasible assembly sequences. This approach will ensure that the generated sequences are always feasible by sorting the nodes into ‘available’ and ‘unavailable’ tasks, during the sequence generation process (Moon et al., 2002).

To test the generated problems, three different algorithms were selected for each problem type. For ASP problems, a multi-objective genetic algorithm (MOGA) as used in Choi et al. (2009) is chosen. This algorithm is selected because, in common with this work, it used task-based representation in

representing ASP problems. Additionally, genetic algorithm is one of the most frequently used algorithms for solving and optimising ASP problems (Rashid et al., 2012b). In this algorithm, the fitness function for ASP is as follows.

$$f_1 = \frac{dc}{dc_{max}} + \frac{tc}{tc_{max}}$$

Eq. 5.8

dc – number of direction changes
tc – number of tool changes
dc_{max} – maximum possible number of direction changes
tc_{max} – maximum possible number of tool changes
dc_{max}, tc_{max} – number of nodes – 1

To test the ALB problem, an ant colony optimisation (ACO) algorithm that has been used for simple assembly line balancing problems (SALBP) in Bautista and Pereira (2007) is used. This algorithm is selected based on citation popularity. In addition, ant colony algorithm is also one of the frequently used algorithms used to solve and optimise ALB problems (Rashid et al., 2012b). In this algorithm, the fitness function is designed as follows.

$$f_2 = \frac{ct}{ct_{max}} + \frac{nws}{nws_{max}} + \frac{wload}{wload_{max}}$$

Eq. 5.9

ct – cycle time
nws – number of workstations
$wload$ – workload variance
ct_{max} – maximum possible cycle time
nws_{max} – maximum possible number of workstations
$wload_{max}$ – maximum possible workload variance

Finally, for integrated ASP and ALB problems, a Hybrid Genetic Algorithm (HGA) as used in Chen et al. (2002) is selected. This algorithm is also selected based on the popularity of this work for integrated ASP and ALB. The concept

and description of integrated ASP and ALB problems is presented in Section 6.2. The fitness function for this problem is designed as follows.

$$f_3 = f_1 + f_2$$

$$f_3 = \frac{dc}{dc_{max}} + \frac{tc}{tc_{max}} + \frac{ct}{ct_{max}} + \frac{nws}{nws_{max}} + \frac{wload}{wload_{max}}$$

Eq. 5.10

5.4.2 Phase 2: Algorithm Testing Using Generated Problems

In Phase 2, the algorithms' performance to generate Pareto optimal solution for combined ASP and ALB problem are tested. The purpose of this test is to determine that the problems generated by the TPG have sufficient variety to enable users to perceive differences in algorithm performance. To perform this test, the MOGA and ACO algorithm previously used to optimise ASP and ALB individually will be used to optimise combined ASP and ALB problem alongside Hybrid GA. The objective function set for this experiment is as follows.

f_1 = minimise number of direction changes

f_2 = minimise number of tool changes

f_3 = minimise cycle time

f_4 = minimise number of workstations

f_5 = minimise workload deviation

In order to evaluate the performance of each algorithm when dealing with different complexity problems, the following performance indicators, adopted from Deb (2001) and Yoosefelahi et al.(2012), are used.

i. Number of non-dominated solution in Pareto optimal, $\tilde{\eta}$: Show the number of non-dominated solutions generated by each algorithm in Pareto solution. Higher $\tilde{\eta}$ indicates better algorithm performance.

ii. Error Ratio, *ER*: *ER* is given by dividing the number of solutions which are not members of the Pareto optimal set with the total number of solutions generated by algorithm *q*. Smaller *ER* indicates better algorithm performance.

iii. Generational Distance, *GD*: *GD* finds an average distance of solution with the nearest Pareto optimal solution. Smaller *GD* indicates better algorithm performance.

$$GD_q = \frac{\sum_{i=1}^{s_q} d_i}{s_q}$$

Eq. 5.11

s_q – number of solutions generated by algorithm *q*

$$d_i = \min_{k=1}^{P} \sqrt{\sum_{m=1}^{M} (f_m^{(i)} - f_m^{*(k)})^2}$$

Eq. 5.12

Where $f_m^{(i)}$ is the *m*-th objective function value of solution *i* and $f_m^{*(k)}$ is the *m*-th objective function value of k^{th} member of Pareto optimal set.

iv. Spacing: This indicator measures the relative distance between each solution.

$$Spacing = \sqrt{\frac{1}{N} \sum_{i=1}^{s_q} (d_i - \bar{d})^2}$$

Eq. 5.13

d_i is distance between solution *i* and the nearest solution, while $\bar{d}$ is average of all d_i. Smaller *Spacing* indicates better uniformity of space between solutions.

v. Maximum Spread, $Spread_{max}$: Measures the extent of solution distribution found by the algorithm. Larger maximum spread is better.

$$Spread_{max} = \sqrt{\sum_{i=1}^{M} (\min f_i - \max f_i)^2}$$

Eq. 5.14

5.5 Results and Discussion on Tuneable TPG

5.5.1 Phase 1 Results

The output from Phase 1 experiments are presented in Figures 5.6 to 5.9, showing the average of best fitness value from ten runs.

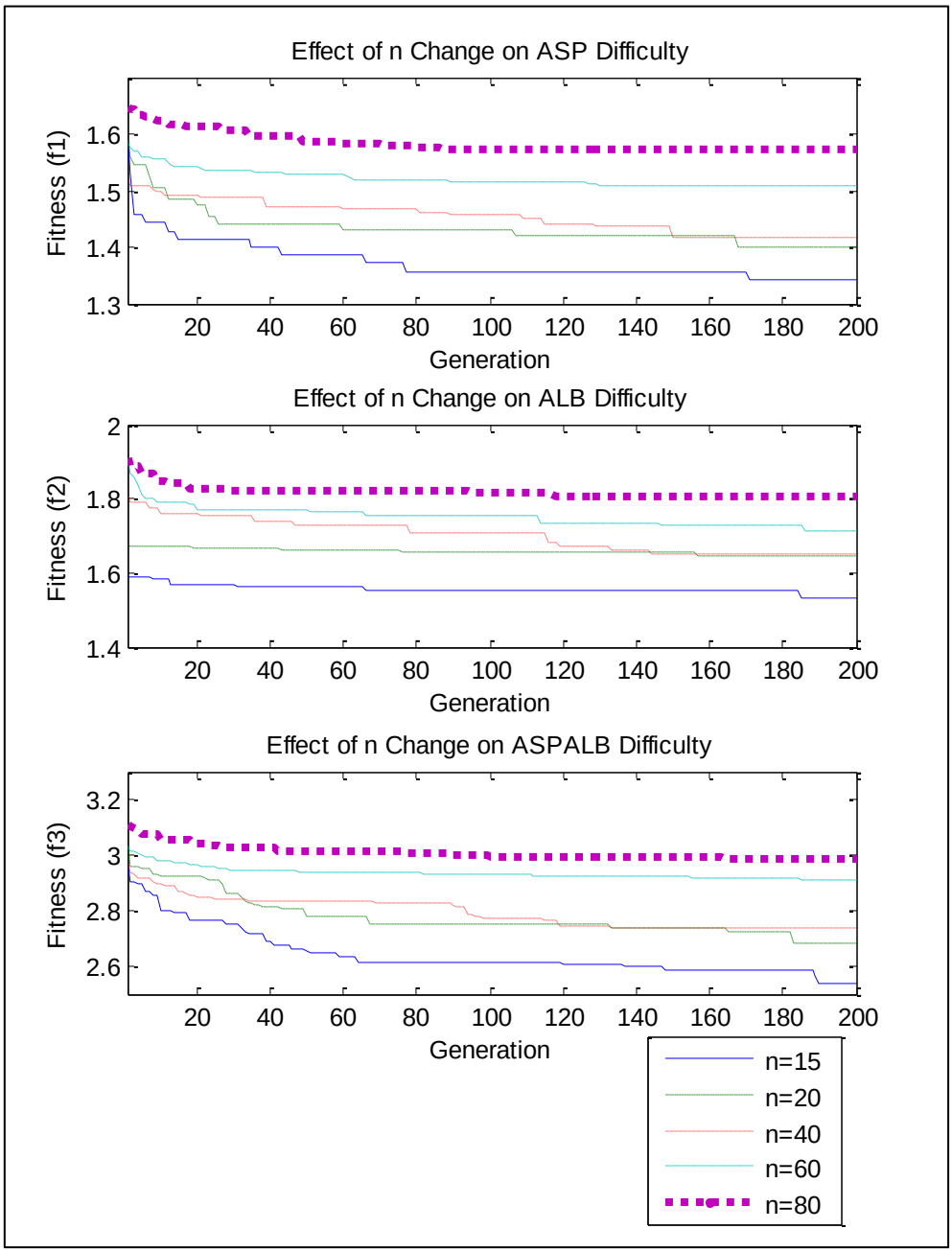


Figure 5.6: Average of best fitness for a range of *n* (number of tasks)

Number of tasks (*n*) Figure 5.6 shows the effect of *n* on the ASP, ALB and integrated ASP and ALB problem difficulties. In all cases, the problems with the larger number of tasks tend to have better fitness although they have similar tuneable input setting for *OS*, *FR* and *TV*. This output pattern is related to the increment of problem difficulties when the number of tasks is increased. The output trend is also consistent with previous works, e.g. Scholl (1993), Bhattacharjee and Sahu (1990) and Otto et al. (2011).

Order Strength (*OS*) Figure 5.7 shows the effect of *OS* change for ASP and ALB problems. In these graphs, the ASP problems with high *OS* values tend to produce better fitness values compared to low and medium *OS* values. A similar output pattern is also found in ALB and integrated ASP and ALB problems, as shown in Figure 5.7. This result indicates that problems with higher *OS* values will have lesser difficulty levels compared to low *OS* values. This finding corroborates a few previous works such as in Mastor (1970), Johnson (1981) Urban and Chiang (2006), while contradicting a few works that associate higher *OS* values with greater complexity (Scholl, 1993; Otto et al., 2011).

This mismatch is due to the dissimilar approaches used in solving the precedence graph. In the works that directly used generated permutation as assembly sequence, precedence graphs with higher *OS* values are harder to solve. Direct permutation has a high probability of generating unfeasible sequences, since the numbers of precedence constraints in high *OS* graphs are higher than low *OS* graph while the search space for both conditions remains the same.

On the other hand, in the works that ensure the feasibility of the sequence such as using topological sort, the precedence graph with higher *OS* is easier to solve, because of differences in search space size. The *OS* value directly influences the number of possible feasible sequence in a precedence graph. In this case, the number of feasible sequences in high *OS* is smaller than in low *OS* because the precedence constraints limit the flexibility of re-sequencing. Since the search space for the precedence graph with high *OS* is smaller than

low *OS*, it is easier to generate solutions with better fitness in high *OS* graphs than with low *OS* graphs.

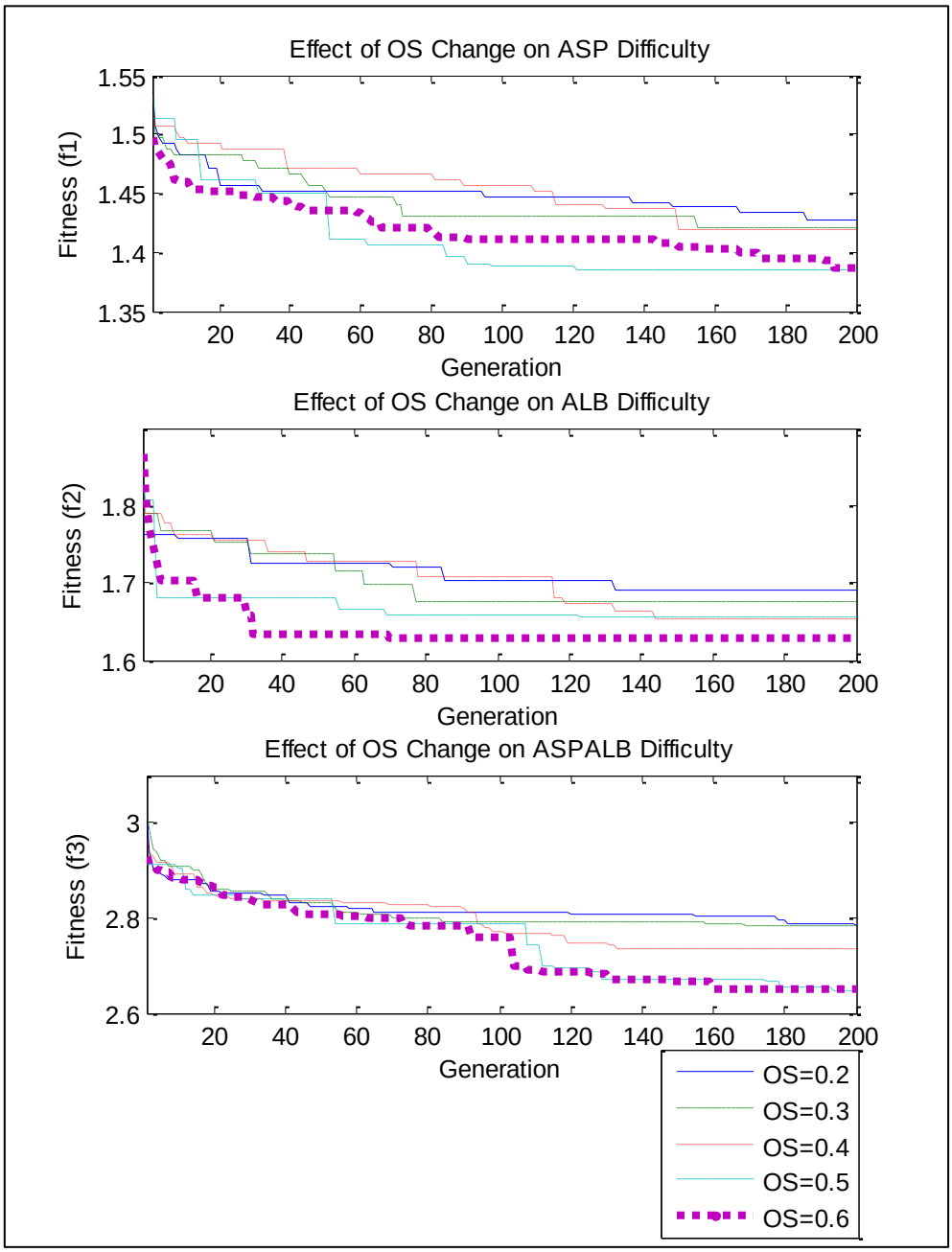


Figure 5.7: Average of best fitness for a range of *OS* value

Nevertheless, there is inconsistency in outputs for ASP with *OS* 0.5 and 0.6, ALB with *OS* 0.4 and 0.5 and integrated ASP and ALB with *OS* 0.5 and 0.6. For these cases, the problem with smaller *OS* emerges with better fitness compared with larger *OS*. A likely explanation is that the chosen *OS* gaps for these

problems are too small, since it does not happen in larger *OS* gaps such as between *OS* 0.6 and 0.4 or smaller. A small *OS* gap means that there is only small search space difference between the two problems which has influenced the inconsistency of results for both conditions. Therefore, to ensure a clear separation between one difficulty levels with another, *OS* gaps which are too small should be avoided.

Frequency Ratio (*FR*) The output from the ASP problem in Figure 5.8 shows that the proposed complexity attribute *FR* can be used to control the ASP data complexity.

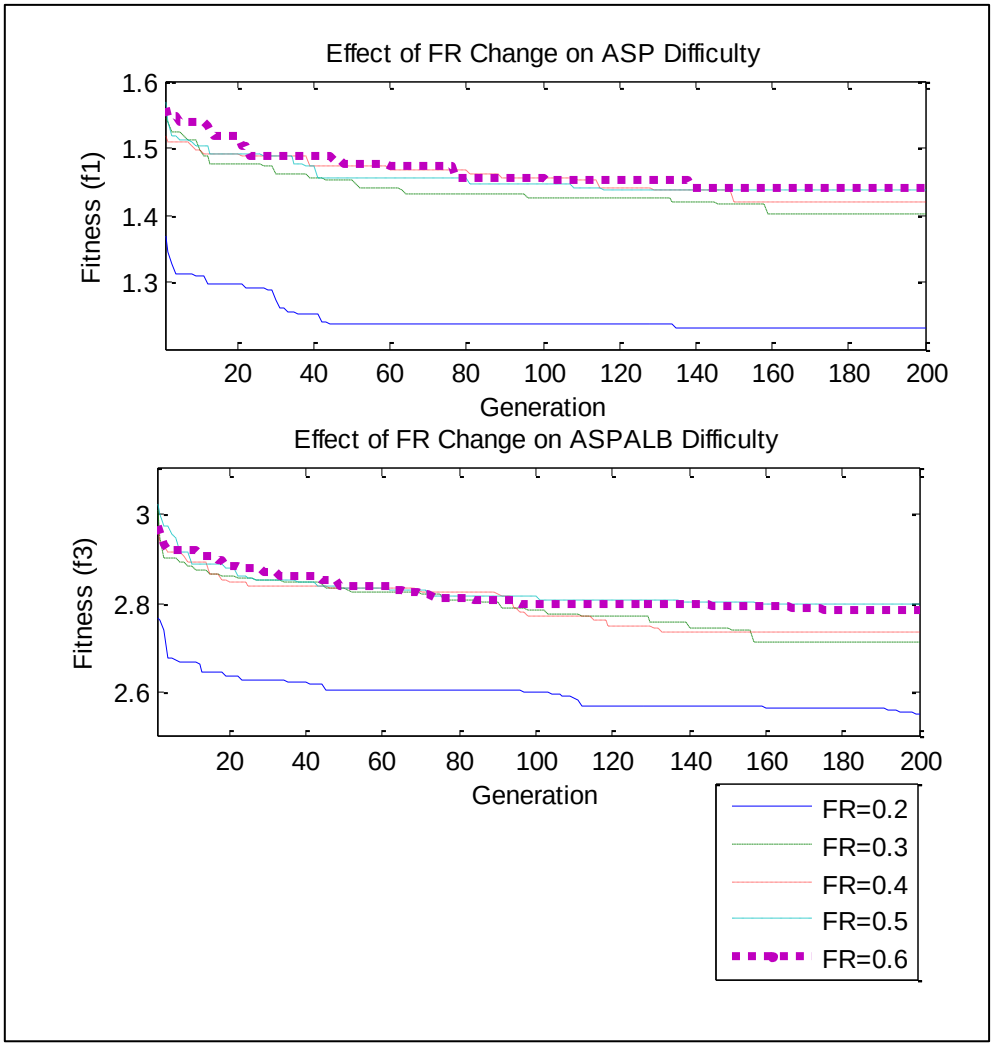


Figure 5.8: Average of best fitness for a range of Frequency Ratio

ASP data with high *FR* will have a wider range of choices that directly increases the size of search space. In contrast, ASP data with low *FR* has smaller search space due to a more limited data variety. As a consequence, the algorithms found it more difficult to achieve minimum direction and tool change for ASP data with higher *FR*.

Time Variability ratio (*TV*) ALB results in Figure 5.9 confirm that the Time Variability ratio (*TV*) adopted from previous works is effective in controlling the assembly time data complexity (Scholl, 1993; Otto et al., 2011).

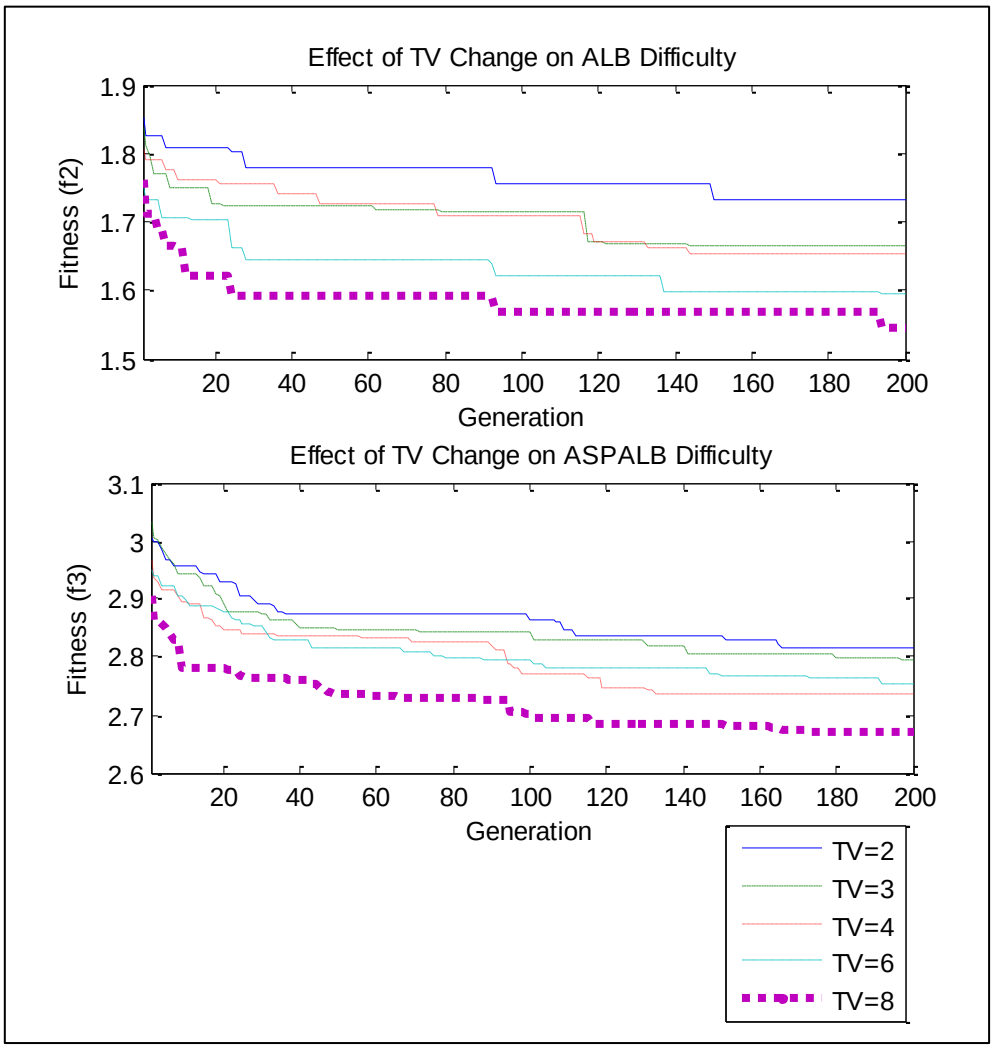


Figure 5.9: Average of best fitness for a range of Time Variability ratio

The assembly times with higher *TV* are easier to arrange because the combination of small and large task times tend to fit the cycle time better than

uniformly large task times (low *TV*). Finally, the integrated ASP and ALB outputs in this figure clearly show that the *TV* input variable is able to control the assembly data difficulties as expected.

The results of the tuneable input test show that ASP and ALB problem complexity can be controlled via the input attributes of the test problem generator. Although the early assumption that the precedence graph with higher *OS* will have greater complexity is unfounded, this attribute's usefulness is maintained by redefining its value: to generate precedence graphs with low complexity, higher *OS* level must be used, while for graphs with high complexity, the *OS* must be set to the lower level. It is found that the selection of tuneable input level is also important to ensure that the desired problem difficulty is achieved. Selection of proper gaps between one level and another is very important to avoid inconsistent problem difficulty.

In order to test the significance of the results, statistical tests are performed. In this case, ANOVA test is carried out to test whether there are any significant differences between the results of one level with results from another level. The null hypothesis stated that there would be no difference between five tuneable input levels means. The summary of the ANOVA test is presented in Table 5.6.

Table 5.6: Summary of ANOVA test

	n change	*OS* change	*FR* change	*TV* change
SSB	16.4897	0.9418	7.3437	2.5119
SSW	3.2268	3.7542	2.1810	2.1205
SST	19.7165	4.6961	9.5247	4.6324
MSB	4.1224	0.2354	1.8359	0.6280
MSW	0.0032	0.0037	0.0021	0.0021
f	1288.25	63.620	874.23	299.05

SSB – Sum of square between groups

SSW – Sum of square within groups

SST – Sum of square total

MSB – Mean squares between groups

MSW – Mean squares within groups

In this case, the critical *f*-value (*f**) that is acquired with 0.05 level of significance from *f*-distribution table is 2.38 (Coolidge, 2000). Table 5.6 consistently shows larger *f*-values compared to *f**. Since all the *f*-values are larger than *f**, the null hypothesis for all tuneable inputs are rejected. In other words, it shows that at 0.05 confidence levels, there are statistically significant differences between levels for *n*, *OS*, *FR* and *TV*.

However, this test does not tell us the exact groups or levels that have statistically significant difference in means. Therefore, an *a posteriori* test known as Tukey's Honestly Significant Different test (Tukey's HSD test) is performed.

Tukey's HSD test compares the mean of rejected null hypothesis with the means of other groups to identify whether there is any significant difference between the mean of one level and another. The value of the absolute difference between two means will be compared to a critical HSD as proposed in the Tukey's table (Coolidge, 2000). The summary of Tukey's HSD test at 0.05 confidence interval is presented in Table 5.7.

From Table 5.7, the absolute mean difference between two levels for *n* and *TV* consistently indicates larger values than the critical HSD value. It shows that there are significant differences in all levels for *n* and *TV*. It means that the problem difficulties for these variables can be statistically distinguished between each level.

Meanwhile, in *OS* and *FR* variables, the absolute mean difference also shows larger values than critical HSD except for the cases between *OS* values of 0.2 and 0.3, *OS* values 0.5 and 0.6, *FR* values 0.3 and 0.4, and *FR* values 0.5 and 0.6. This result is related to the selection of appropriate gaps between levels, since it only occurs between adjacent levels. Consistent with earlier discussion on the effect of *OS* change on the problem difficulties (Figure 5.7), too small gaps between consecutive levels should be avoided.

Table 5.7: Summary of Tukey's HSD test

Variable (critical HSD)	**Comparison Level**		**Absolute Mean Difference**
n (0.015536)	15	20	0.1331
	15	40	0.1497
	15	60	0.2929
	15	80	0.3658
	20	40	0.0166
	20	60	0.1598
	20	80	0.2327
	40	60	0.1432
	40	80	0.2161
	60	80	0.0729
OS (0.016759)	0.2	0.3	0.0080
	0.2	0.4	0.0292
	0.2	0.5	0.0680
	0.2	0.6	0.0755
	0.3	0.4	0.0212
	0.3	0.5	0.0600
	0.3	0.6	0.0675
	0.4	0.5	0.0388
	0.4	0.6	0.0169
	0.5	0.6	0.0075
FR (0.012773)	0.2	0.3	0.1920
	0.2	0.4	0.1954
	0.2	0.5	0.2301
	0.2	0.6	0.2256
	0.3	0.4	0.0034
	0.3	0.5	0.0381
	0.3	0.6	0.0336
	0.4	0.5	0.0347
	0.4	0.6	0.0302
	0.5	0.6	0.0045
TV (0.009053)	2	3	0.0205
	2	4	0.0708
	2	6	0.0894
	2	8	0.1453
	3	4	0.0503
	3	6	0.0389
	3	8	0.1248
	4	6	0.0114
	4	8	0.0745
	6	8	0.0859

5.5.2 Phase 2 Results

In this phase, 25 test problems with different difficulty settings are used to demonstrate the usefulness of the TPG for testing the performance of algorithms. The assembly problem for this experiment is set up as in Table 5.8 and Table 5.9.

Table 5.8: Problem setting for experiments in Phase 2

Problem	Graph Difficulties	Data Variables
1	Low	Low
2	Low	Low-Med
3	Low	Medium
4	Low	Med-High
5	Low	High
6	Low-Med	Low
7	Low-Med	Low-Med
8	Low-Med	Medium
9	Low-Med	Med-High
10	Low-Med	High
11	Medium	Low
12	Medium	Low-Med
13	Medium	Medium
14	Medium	Med-High
15	Medium	High
16	Med-High	Low
17	Med-High	Low-Med
18	Med-High	Medium
19	Med-High	Med-High
20	Med-High	High
21	High	Low
22	High	Low-Med
23	High	Medium
24	High	Med-High
25	High	High

Table 5.9: Attribute settings for different graph and data difficulty levels

Level	**Graph Difficulty**		**Data Difficulty**	
	n	***OS***	***FR***	***TV***
Low	15	0.6	0.2	8
Low-Med	20	0.5	0.3	6
Med	40	0.4	0.4	4
Med-High	60	0.3	0.5	3
High	80	0.2	0.6	2

The tuneable input for these test problems are grouped into Graph Difficulty (*n* and *OS* variables) and Data Difficulty (*FR* and *TV* variables).

The results from Phase 2 experiments are summarised in Table 5.10. Numbers in brackets are weighting values that are assigned to each algorithm based on its performance for the respective indicator. For every indicator in a given problem, the best result is assigned weight value 3, while the second and third positions are assigned weight values 2 and 1 respectively. Then, the algorithm ranking is made through comparison of the weighted sums.

Table 5.10: Summary of the result of experiments on selected multi-objective algorithms

*Numbers in brackets are weighting values (*W*) from the best (weight=3) to worst (weight=1) performance.

Problem	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$	Σ *W*	Rank
1	MOGA	15(2)	0.5946(2)	1.0340(1)	1.2913(2)	34.2251(2)	9	2
	ACO	12(1)	0.6250(1)	0.9694(2)	1.1781(3)	34.1030(1)	8	3
	HGA	24(3)	0.2727(3)	0.3565(3)	2.2112(1)	36.9763(3)	13	1
2	MOGA	22(2)	0.4359(2)	0.7180(2)	1.0435(2)	34.2097(3)	11	2
	ACO	11(1)	0.6452(1)	1.1590(1)	1.4938(1)	33.3108(2)	6	3
	HGA	35(3)	0.1667(3)	0.2681(3)	0.8956(3)	31.2778(1)	13	1

Problem	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$	Σ *W*	Rank
3	MOGA	12(2)	0.7447(1)	1.1370(2)	1.0293(3)	40.2682(2)	10	2
	ACO	10(1)	0.6429(2)	1.1764(1)	1.8612(1)	38.1782(1)	6	3
	HGA	40(3)	0.1489(3)	0.2517(3)	1.4235(2)	41.1657(3)	14	1
4	MOGA	29(2)	0.3556(1)	0.4950(2)	1.3678(2)	36.5902(1)	8	3
	ACO	26(1)	0.2973(3)	0.4366(3)	1.7489(1)	38.3385(2)	10	2
	HGA	35(3)	0.3519(2)	0.5074(1)	1.1018(3)	38.5984(3)	12	1
5	MOGA	11(2)	0.5926(2)	0.9460(2)	1.2987(3)	33.5636(2)	11	2
	ACO	10(1)	0.6429(1)	1.0924(1)	1.3385(2)	34.3586(3)	8	3
	HGA	22(3)	0.2667(3)	0.3276(3)	1.6657(1)	34.3586(3)	13	1
6	MOGA	13(1)	0.7869(1)	1.7381(1)	1.6819(1)	39.5870(1)	5	3
	ACO	25(3)	0.5098(2)	1.3454(2)	1.6183(2)	39.6918(2)	10	2
	HGA	40(2)	0.4030(3)	0.6576(3)	1.4541(3)	40.0436(3)	15	1
7	MOGA	16(2)	0.6098(1)	1.1300(1)	1.6974(1)	37.2650(1)	6	3
	ACO	16(2)	0.5789(2)	1.0099(2)	1.5133(2)	39.3564(3)	11	2
	HGA	41(3)	0.3051(3)	0.5166(3)	1.1344(3)	39.0404(2)	14	1
8	MOGA	17(2)	0.7018(1)	1.1665(1)	1.3567(2)	39.0331(3)	9	2
	ACO	16(1)	0.5429(2)	1.0410(2)	1.8336(1)	37.3966(1)	7	3
	HGA	40(3)	0.3443(3)	0.5396(3)	0.9635(3)	37.7713(2)	14	1
9	MOGA	17(2)	0.6909(1)	1.1311(1)	1.1018(3)	39.7311(2)	9	2
	ACO	14(1)	0.5882(2)	1.0600(2)	1.6150(2)	38.1256(1)	8	3
	HGA	36(3)	0.4930(3)	0.8795(3)	1.7604(1)	44.9119(3)	13	1
10	MOGA	16(1)	0.6667(1)	1.2655(1)	1.5753(1)	36.4029(1)	5	3
	ACO	25(2)	0.5763(2)	1.1164(2)	1.3033(2)	37.8426(3)	11	2
	HGA	30(3)	0.5238(3)	0.7406(3)	1.0246(3)	37.6970(2)	14	1
11	MOGA	17(1)	0.8496(1)	1.9129(1)	1.3149(3)	45.1158(1)	7	3
	ACO	60(2)	0.5000(2)	0.9915(2)	1.3560(2)	46.4900(3)	11	2
	HGA	86(3)	0.3723(3)	0.7732(3)	1.4197(1)	45.4979(2)	12	1
12	MOGA	24(1)	0.7073(1)	1.7056(1)	1.8150(1)	48.4041(3)	7	3
	ACO	56(3)	0.3253(3)	0.6771(3)	1.4082(3)	46.8911(1)	13	1
	HGA	44(2)	0.6271(2)	1.3069(2)	1.4886(2)	47.8308(2)	10	2
13	MOGA	20(1)	0.7701(1)	1.6837(1)	1.6846(2)	48.4191(2)	7	3
	ACO	42(2)	0.3731(2)	0.7612(3)	1.8370(1)	47.0473(1)	10	2
	HGA	74(3)	0.4351(3)	0.8760(2)	1.3466(3)	49.9341(3)	13	1
14	MOGA	56(2)	0.3778(3)	0.8536(2)	1.7669(1)	53.7493(1)	9	2
	ACO	65(3)	0.4348(2)	0.8321(3)	1.3163(2)	54.6819(3)	13	1
	HGA	49(1)	0.6797(1)	1.4087(1)	1.1580(3)	53.8050(2)	8	3

Problem	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$	Σ *W*	Rank
15	MOGA	15(1)	0.7857(1)	2.0094(1)	1.5541(3)	50.1276(3)	9	2
	ACO	31(2)	0.5441(2)	1.1165(2)	2.2496(1)	49.3039(1)	8	3
	HGA	49(3)	0.3194(3)	0.7035(3)	1.6625(2)	49.6062(2)	13	1
16	MOGA	38(1)	0.6696(1)	1.6155(1)	1.5986(2)	55.3575(3)	8	3
	ACO	86(3)	0.3723(3)	0.7880(3)	1.7949(1)	55.2207(2)	12	1
	HGA	69(2)	0.6124(2)	1.4092(2)	1.2798(3)	54.1806(1)	10	2
17	MOGA	30(1)	0.7391(1)	1.9710(1)	1.4983(3)	55.7943(3)	9	2
	ACO	62(2)	0.5000(3)	1.2519(2)	1.6083(1)	55.1087(1)	9	2
	HGA	73(3)	0.5494(2)	1.2063(3)	1.5393(2)	55.7024(2)	12	1
18	MOGA	22(1)	0.8182(1)	1.9982(1)	1.6087(2)	57.2777(1)	6	3
	ACO	88(3)	0.2000(3)	0.5709(3)	2.0579(1)	59.0822(3)	13	1
	HGA	74(2)	0.5747(2)	1.4657(2)	1.4820(3)	57.3599(2)	11	2
19	MOGA	42(1)	0.5922(1)	1.6303(1)	1.6454(3)	57.3484(2)	8	3
	ACO	49(2)	0.5664(2)	1.4639(2)	1.6960(2)	57.1157(1)	9	2
	HGA	74(3)	0.4559(3)	1.0453(3)	1.8645(1)	58.5914(3)	13	1
20	MOGA	33(1)	0.6972(1)	1.5315(1)	1.6453(2)	61.8964(2)	7	3
	ACO	66(2)	0.4000(3)	0.9788(2)	1.9603(1)	61.3480(1)	9	2
	HGA	84(3)	0.4650(2)	0.8983(3)	1.4834(3)	63.2256(3)	14	1
21	MOGA	17(1)	0.8411(1)	2.2525(1)	1.4918(2)	56.7749(1)	6	3
	ACO	117(3)	0.1761(3)	0.3228(3)	1.5736(1)	58.4656(3)	13	1
	HGA	57(2)	0.6780(2)	1.7245(2)	1.4484(3)	58.2365(2)	11	2
22	MOGA	44(1)	0.6944(1)	1.6900(1)	1.8500(2)	62.3752(2)	7	3
	ACO	90(3)	0.2857(3)	0.8241(3)	1.9158(1)	65.5746(3)	13	1
	HGA	70(2)	0.6410(2)	1.5881(2)	1.3136(3)	62.1336(1)	10	2
23	MOGA	21(1)	0.8397(1)	2.4638(1)	1.8718(2)	63.7124(2)	7	3
	ACO	46(2)	0.4458(3)	1.1937(3)	1.9034(1)	64.4027(3)	12	1
	HGA	74(3)	0.5912(2)	1.9811(2)	1.6302(3)	63.6888(1)	11	2
24	MOGA	39(1)	0.6286(2)	1.6486(1)	1.8588(2)	64.5943(1)	7	3
	ACO	71(3)	0.3238(3)	0.8033(3)	3.2717(1)	67.5944(3)	13	1
	HGA	61(2)	0.6494(1)	1.6160(2)	1.5992(3)	65.5225(2)	10	2
25	MOGA	57(1)	0.7077(2)	2.1692(2)	1.7039(2)	72.1097(1)	8	3
	ACO	105(3)	0.2606(3)	0.5954(3)	2.3302(1)	73.1415(3)	13	1
	HGA	74(2)	0.7218(1)	2.3913(1)	1.6459(3)	72.5640(2)	9	2

Based on the results in Table 5.10, the HGA consistently show the best performance in all problems, having low and low-medium graph difficulties

(problems 1-10). Meanwhile, the MOGA shows better performance compared to the ACO algorithm for problems 1-3, but then showed inconsistent performance for problems 4 to 10. In problems 4 to 10, the ACO algorithm starts to overcome the MOGA performance in some cases.

Meanwhile, for the problems with medium and medium-high graph difficulties (problems 11 – 20), the HGA and ACO algorithms alternately lead the algorithms in the first rank. However, when the graph difficulty is increased to high difficulty (problems 21 – 25), ACO has consistently shows better performance and then followed by HGA and MOGA. The relative performance of each algorithm is presented graphically in Figure 5.10.

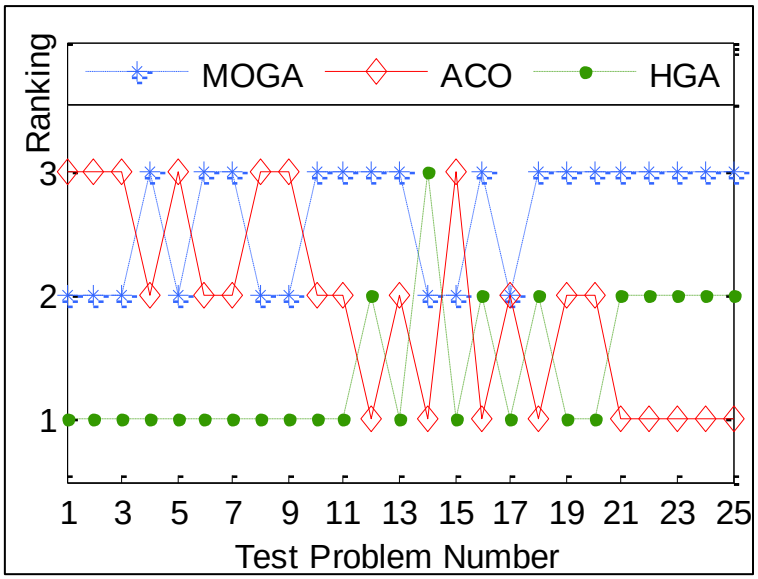


Figure 5.10: Algorithm's ranking for the range of test problems

5.6 Chapter Summary

In this chapter, a test problem generator (TPG) with tuneable complexity for ASP and ALB problems has been proposed. A set of experiments has been conducted to assess the TPG. Experimental results confirm that problem complexities can be controlled by tuneable input variables.

The results from Phase 1 experiments which test the effects of tuneable inputs confirm the ability of TPG to generate problem with varying complexity levels. The problem difficulties will increase when using a larger number of tasks (*n*), smaller Order Strength (*OS*) value, larger Frequency Ratio (*FR*) or smaller Time Variability ratio (*TV*). As presented in the results, the *n* and *OS* influence the assembly graph difficulties, while *FR* and *TV* influence the assembly data difficulties. The result of statistical tests confirmed that there are significant differences in the problem difficulties when changing the value of *n* and *TV* variables. On the other hand, the significant difference of problem difficulties can also be achieved by selecting appropriate values for *OS* and *FR,* as suggested in Table 5.1.

One of important finding in this chapter is the contradiction of *OS* effect between experiment and some of papers from the literature. This is because of different approaches used to handle precedence constraint. For the approach that uses repair mechanism to ensure assembly sequence feasibility (as used in this work), the problem with larger *OS* is easier to optimise because this approach has reduced the search space to only feasible region. While for the approach that does not guarantee sequence feasibility, larger *OS* is harder to optimise because it tends to generate infeasible assembly sequences.

Results of the algorithm performance experiment in Phase 2 show that the Hybrid Genetic Algorithm (HGA) consistently performed well in optimising problems with low and medium difficulties. Meanwhile, the Ant Colony Optimisation (ACO) showed good performance in problems with high level of difficulty. Based on the performance of both algorithms, the HGA is recommended for integrated ASP and ALB problems with low and medium

difficulties, whilst the ACO is for ASP and ALB problems at high difficulty. These findings confirm that the problems generated by the TPG offer sufficient range of problem variety to be used in algorithm testing. The generated problems were found to be useful to identify the strengths and weaknesses of the tested algorithms.

Although further experiments are needed to confirm these strengths and weaknesses, TPG has provided an important path by supplying a variety of ASP and ALB problems for systematic testing. Therefore, it can be concluded that the proposed test problem generator is able to generate combined ASP and ALB problems in a wide range of difficulties.

In summary, this chapter has achieved the following goals:

- The requirements and specifications for the proposed TPG have been explained.
- The methodology of the TPG development and example of application has been explained.
- The experimental design to test the proposed TPG for ASP and ALB has been described.
- The finding from experiment of the tuneable test problem generator has been discussed.

CHAPTER 6

DEVELOPMENT OF MULTI-OBJECTIVE DISCRETE PARTICLE SWARM OPTIMISATION ALGORITHM

This chapter proposes an algorithm to optimise integrated ASP and ALB problems, named Multi-Objective Discrete Particle Swarm Optimisation (MODPSO) algorithm. The algorithm development methodology that consists of descriptions of objective functions, problem representation, constraint-handling and the proposed MODPSO algorithm will be explained. Then, the experimental strategy to test the algorithm and also performance indicators for optimisation algorithms which seek non-dominated solutions also will be discussed. Next, the discussion of experimental results which deploy various algorithms to optimise multi-objective ASP and ALB problems will be presented, followed by the conclusion of the proposed MODPSO algorithm.

This chapter aims to achieve the following goals:

- Present the proposed MODPSO algorithm.
- Explain the experimental design to test the MODPSO algorithm using different range of problem difficulties.
- Present and discuss the experimental results of MODPSO algorithm.

6.1 Background

In recent years, various multi-objective optimisation techniques have been proposed to solve assembly optimisation problems. Recent optimisation work of integrated ASP and ALB, the GA-based algorithms performed well in optimising the problem with low and medium difficulties. However, the performance of GA-based algorithms did not last when optimising high difficulty problems, especially the problems with larger numbers of tasks (Rashid et al., 2012a). In order to overcome the limitation, a new algorithm designed to optimise integrated ASP and ALB problem is needed.

In many different works which compare the performance of algorithms, Particle Swarm Optimisation (PSO) has shown strong performance compared to competing algorithms. This algorithm is popular due to its simplicity and its ability to converge quickly to a reasonably good solution (Xinchao, 2010). PSO is a population-based stochastic optimisation technique that was developed by Kennedy and Eberhart in 1995. In PSO, the potential solutions, named particles, ‘fly’ through the problem space by following the current optimum particles (Kennedy and Eberhart, 1995).

Section 2.4.5 has compared the performance of PSO to GA in individual ASP and ALB optimisation. In the majority of the applications, PSO has shown better performance in terms of computational time and overall performance. In another application, PSO was found to perform better than GA, Memetic Algorithm, Shuffled Frog Leaping and Ant Algorithm in solving continuous and discrete optimisation problems (Elbeltagi et al., 2005).

6.2 Description of Integrated ASP and ALB Optimisation Approach

In current practice, ASP and ALB optimisation are performed sequentially, where the ASP is optimised before the ALB. In this approach, the ASP holds the entire search space that consists of all feasible assembly sequences. The

output from ASP optimisation will be handed over to ALB as the input (refer to Figure 1.3). Figure 6.1 presents the flow of sequential optimisation for ASP and ALB problems.

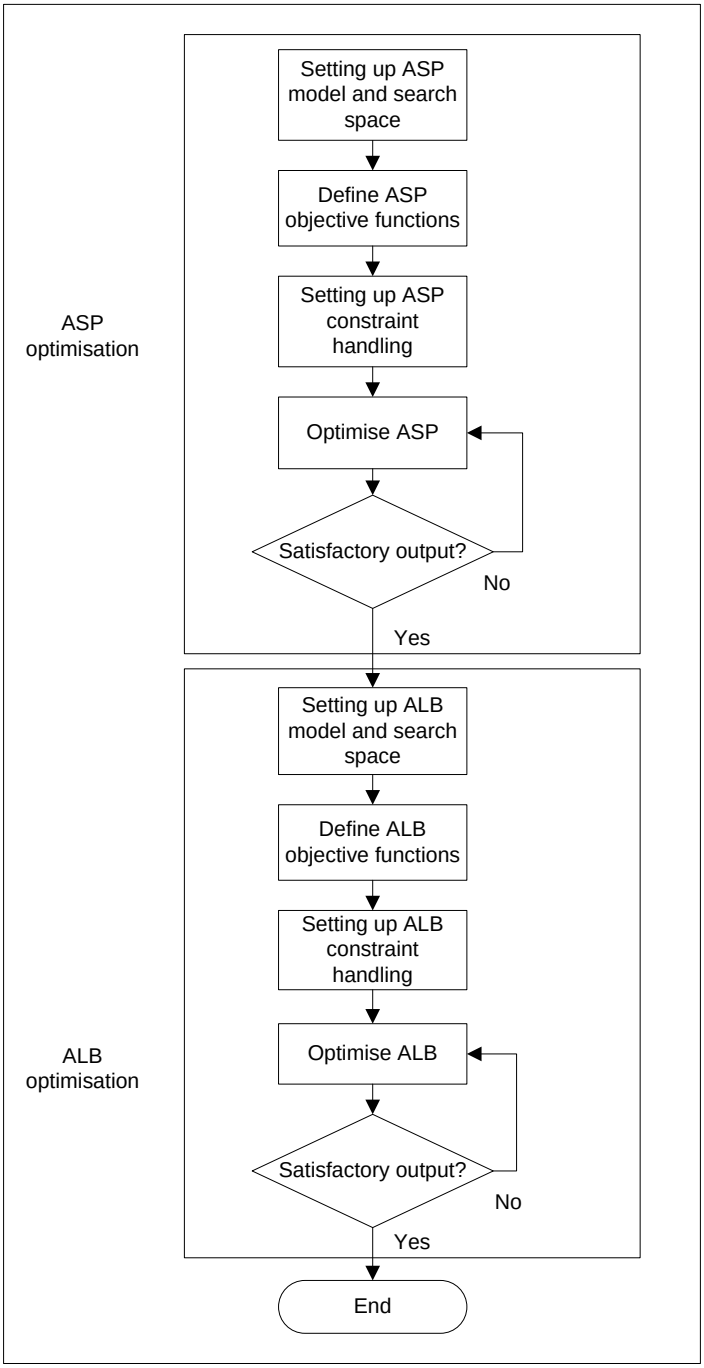


Figure 6.1: Flowchart of sequential ASP and ALB optimisation approach

In the first part of sequential optimisation, the ASP problem is optimised as presented in Figure 6.1. Prior to problem optimisation, the ASP model, objectives functions and constraint handling procedure are set up. Once the ASP optimisation has achieved satisfactory optimum results, the output is handed over to the next stage, where ALB optimisation is taking place. Based on the output from ASP optimisation, the ALB model and design space are defined. Next, the ALB objective functions and constraint handling procedure are set up, before going to the ALB optimisation.

The integrated ASP and ALB optimisation approach used in this thesis optimises both activities concurrently. The flow of this approach is presented in Figure 6.2. In this approach, the ASP and ALB model and design space are set up and defined together. The objective functions and constraint handling procedures are also defined in similar stages for ASP and ALB before the problems are optimised together. A quantitative comparison between sequential and integrated optimisation approaches is presented in Section 8.4.

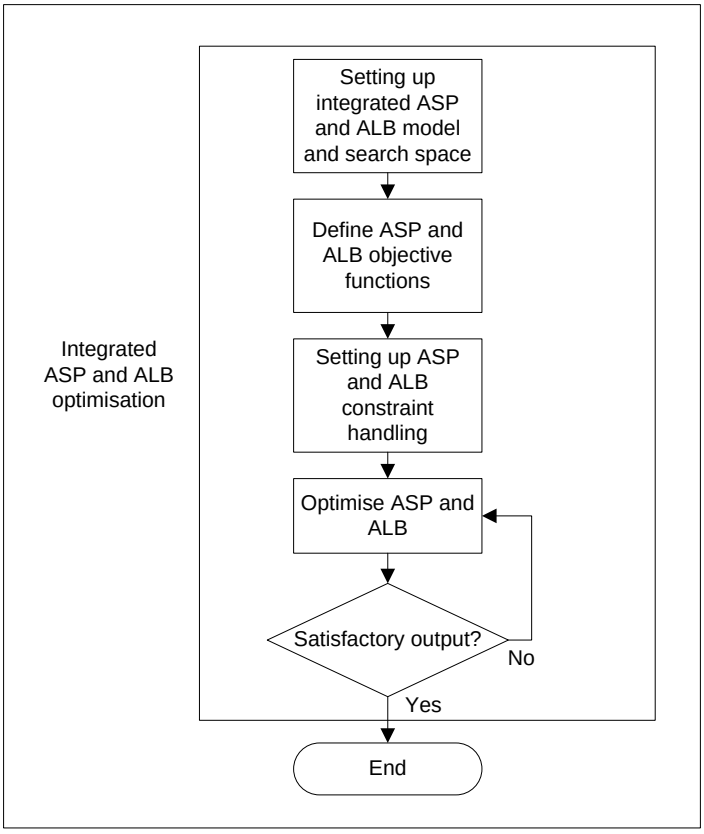


Figure 6.2: Flowchart of integrated ASP and ALB optimisation approach

6.3 MODPSO Algorithm Development Methodology

To develop the proposed algorithm for integrated ASP and ALB optimisation, four major steps will be implemented. The first step is to identify the objective functions for integrated ASP and ALB based on the literature survey. The second step is to develop a representation scheme for integrated problems. The next step is to identify the constraints and design techniques to handle them. The final step is to develop the Multi-Objective Discrete PSO algorithm based on the objective functions, representation and constraints identified in earlier steps.

6.3.1 Objective Function

Various objective functions have been designed and used to optimise ASP and ALB problems. A prior literature survey has collated objective functions that have been used by researchers in both problems (Rashid et al., 2012b). This survey also found that the most frequently used ASP optimisation objectives are to *minimise assembly direction change* and to *minimise number of tool changes*. In ALB works, the dominant optimisation objectives are to *minimise cycle time*, *minimise number of workstations* and *minimise workload variance* (Rashid et al., 2012b).

Number of assembly direction change (n_{dc}) is summation of direction changes within all workstations.

$$n_{dc} = \sum_{i=1}^{nws} \sum_{n=2}^{N_i} DC_n$$

Eq. 6.1[2]

For *nws* is the number of workstations and N_i is the number of task in i^{th} workstation, the direction change for n^{th} task in i^{th} workstation (DC_n) is as follows:

$$DC_n = \begin{cases} 0, & \text{if the assembly direction of } n^{th} \text{ task is similar to the previous task} \\ 1, & \text{if the assembly direction of } n^{th} \text{ task is not the same as the previous task} \end{cases}$$

[2]Eq. 6.1 is repeated from Eq. 4.2

Number of assembly tool change (n_{tc}) is summation of assembly tool changes within all workstations.

$$n_{tc} = \sum_{i=1}^{nws} \sum_{n=2}^{N_i} TC_n$$

Eq. 6.2[3]

$$TC_n = \begin{cases} 0, & \text{if the assembly tool of } n^{\text{th}} \text{ task is similar to the previous task} \\ 1, & \text{if the assembly tool of } n^{\text{th}} \text{ task is not the same as the previous task} \end{cases}$$

Cycle time (*ct*) is the time intervals at which product units must be finished in order to meet demand (Whitney, 2004). In this case, cycle time for particular assembly sequence is the highest processing time among all workstations. Processing time (*pt*) refers to total assembly time in a particular workstation. Once the total processing time for the current workstation is larger than the maximum allowable cycle time (ct_{max}), the present assembly task will be assigned into the next workstation. Normally, ct_{max} is determined from number of demand or required output in the assignment period.

Number of workstation (*nws*) can be determined after all the assembly tasks have been assigned into workstations. Once completed, the number of the generated workstation is used as the fourth objective. The number of workstation depends on cycle time where larger cycle time leads to smaller number of workstation and vice versa.

Workload variation (*v*) calculates the average amount of idle time in workstations. In this case, the smaller the workload variation shows that the assembly line have an almost equal load between workstations.

$$v = \frac{\sum_{i=1}^{nws} (ct\text{-}pt_i)}{nws}$$

Eq. 6.3[4]

[3] Eq. 6.2 is repeated from Eq. 4.3
[4] Eq. 6.3 is repeated from Eq. 2.5

6.3.2 ASP and ALB Problem Representation

In order to incorporate ASP and ALB optimisations into a single integrated optimisation, a clear prerequisite is the availability of an integrated ASP and ALB representation. For this purpose, an integrated assembly task-based representation scheme, as presented in Chapter 4, is used to represent both ASP and ALB problems. As mentioned in Chapter 4, the assembly plan is represented by a *precedence graph* (Figure 6.3) and the assembly data is presented in a *data table* (Table 6.1). Each node in the precedence graph represents an assembly task, while the connecting arc represents assembly precedence.

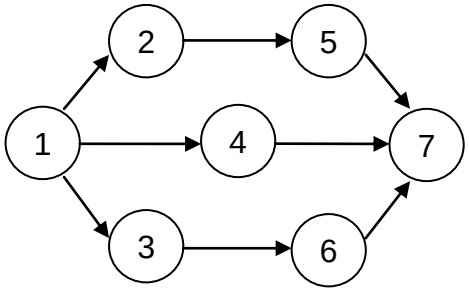


Figure 6.3: Example of precedence graph

Table 6.1: Data table

Task	*Direction*	*Tool*	*Time*
1	*+x*	*T1*	4
2	*-x*	*T2*	12
3	*+x*	*T1*	7
4	*-x*	*T3*	4
5	*+x*	*T1*	12
6	*+x*	*T1*	5
7	*-x*	*T2*	12

For computational purposes the precedence graph is transformed into a precedence matrix. The precedence matrix in Table 6.2 represents the assembly precedence graph in Figure 6.3. In this matrix, the value 0 shows no precedence relation between task *i* and task *j*, while the value 1 shows that the task *i* must be performed prior to task *j*.

Table 6.2: Precedence matrix

i	*j*						
	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	1	1	1	0	0	0
2	0	0	0	0	1	0	0
3	0	0	0	0	0	1	0
4	0	0	0	0	0	0	1
5	0	0	0	0	0	0	1
6	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0

6.3.3 Precedence Constraint Handling

The main constraint in this work is *precedence restriction,* which represents the compulsory sequence that must be followed when assembling a particular product. In handling this constraint, the topological sort approach is applied. Topological sort is an approach designed to establish feasible sequence by selecting only one available assembly task in each iteration. The topological sort procedure is repeated until all tasks are selected (Moon et al., 2002).

Procedure: Topological Sort
Begin
n; number of tasks
st = 0; number of selected task
While*st* ≤ *n*
- Establish *available set*
- *st* = *st* +1
- Select one task from *available set* and place in st^{th} position of feasible sequence
- Remove all outgoing arcs from selected task
- Eliminate selected task from precedence graph

End While
End Procedure

In the procedure above, *available set* consists of tasks without an incoming arc. Then, one of the tasks in *available set* is selected using a predetermined

selection rule. There are a few selection rules which are regularly used, such as random selection, weight-based selection and ordered-based selection. Next, all the outgoing arcs from the selected task are removed and the selected task is eliminated from the graph to remove the possibility of that task being re-selected.

6.3.4 Proposed Multi-Objective Discrete PSO (MODPSO)

Various versions of PSO algorithms have been proposed to optimise multi-objective problems for individual ASP and ALB (Lv and Lu, 2010; Yu et al., 2009; Nearchou, 2011; Lv et al., 2010; Tseng et al., 2010b). One of the PSO versions for optimising multi-objective problems is known as Discrete PSO (DPSO), first proposed by Rameshkumar et al. (2005) for scheduling problems and later adopted to optimise ASP problem (Lv et al., 2010). However, in these works, the multi-objective problem is handled by bundling all objectives into a single objective that leads to only one solution. This approach required high-quality prior knowledge and experience concerning the importance of an objective compared to others. Another PSO version called Multi-Objective PSO (MOPSO) was proposed by Coello and Lechuga with the objective of extending the application of PSO for multi-objective problems (Coello Coello and Lechuga, 2002). This algorithm uses the original PSO operators for generating new particles position and velocity, but used the non-dominated approach to find the set of optimum solutions.

In order to treat the problem as real multi-objective optimisation, this work proposed to apply the Pareto-based approach in the proposed algorithm. In PSO, the potential solution is represented by a particle, which bring three important vectors; particle position (X_i), particle velocity (V_i) and particle best solutions (*Pbest*). Figure 6.4 shows the working flow of Multi-Objective DPSO (MODPSO) algorithm.

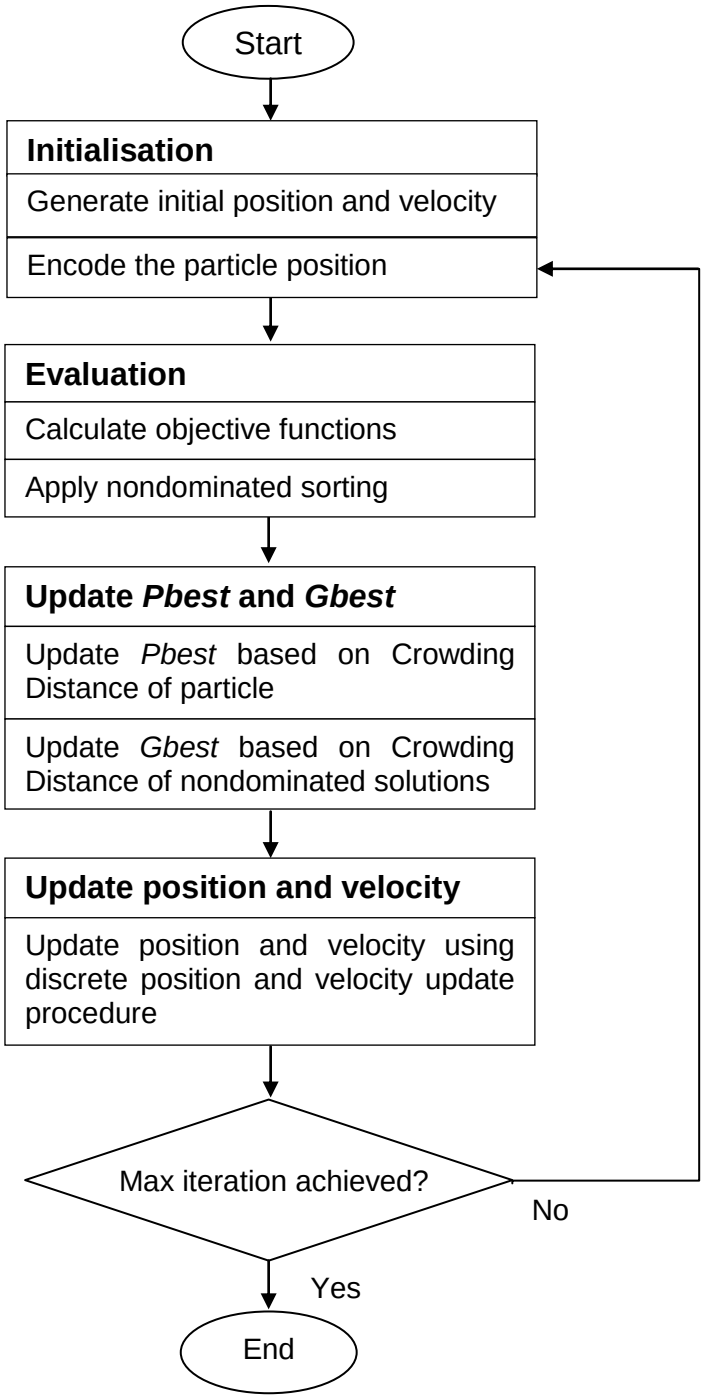


Figure 6.4: Flow chart of the MODPSO

6.3.4.1 Initialisation

The number of particles (*npar*) and maximum number of iterations ($iter_{max}$) is set in this step. Then the initial population which is known as the swarm is produced by generating *npar* set of initial position (*X*) and velocity (*V*) which consists of a permutation of integer from 1 to *n* in random orders. As an example of a precedence graph in Figure 6.3, one of the particles from the initial population is, X_1= [6 3 5 7 1 4 2] and V_1= [2 1 5 6 3 7 4]. Next, the swarm is encoded to generate feasible sequences according to precedence constraint. The swarm is encoded using topological sort procedure, as explained earlier.

In this work, the selection rule for topological sort is set to follow ordered-based selection. It means that the first available task found in the particle order will be selected to be placed in feasible sequence. For the X_1 above, the feasible sequence is established as follows.

Table 6.3: Particle encoding

Iteration	Available set	Selected task
1	1	1
2	2, 3, 4	3
3	2, 4, 6	6
4	2, 4	4
5	2	2
6	5	5
7	7	7

For the first iteration in Table 6.3, the only available task according to Figure 6.3 is task 1, so this task is selected and placed in the first position of feasible sequence. Then, in the second iteration, the available tasks are 2, 3 and 4. By following the ordered-based selection, the first available task found in X_1 is sub-particle 3, so task 3 is selected in the second iteration. This procedure is repeated until all the tasks are selected. The feasible assembly sequence that encoded from X_1 is $fseq_1$= [1 3 6 4 2 5 7].

6.3.4.2 Evaluation

In this step, the encoded feasible sequence is evaluated by using predefined objective functions. The objective functions are calculated using procedures and formulas in Eq. 6.1 –6.3. For given ct_{max} = 17 time unit, the objective functions for feasible sequence $fseq_1$ are as follows:

n_{dc} = 0
n_{tc} = 1
ct = 16 time unit
nws = 4 workstations
v = 2 time unit/workstation

Next, the non-dominated sorting is applied to establish the Pareto optimal solution. This approach is adopted from Deb (2001). The Pareto set is updated in every iteration by evaluating each particle with solution in the Pareto optimal.

6.3.4.3 Update Pbest and Gbest

Pbest is the best personal particle solution, while *Gbest* is the best solution among all particles. To evaluate and determine the *Pbest* and *Gbest*, a mechanism to select the best solution within all particles is needed. For this purpose, Crowding Distance (*CD*) that provides the estimation of solutions density surrounding that solution is used. For *Pbest*, the *CD* is calculated within solution in swarm, whereas the *CD* for *Gbest* is calculated within the Pareto set. The following algorithm is used to calculate the *CD* of each point in the set *R*(Deb, 2001).

Crowding Distance Calculation Procedure:

Step 1 Call the number of solution in *R* as l = $\left|R\right|$. For each *i* in the set, first assign d_i = 0.

Step 2 For each objective function m = 1, 2,…,*M*, sort the set in descending order of r_m.

Step 3 For m = 1, 2,…,M, assign maximum (max_m) and minimum (min_m) value for each objectives m.

Step 4 Calculate d_i^m for each of objective m for solution i.

$$d_i^m = \left(\frac{I_{up^i}^m - I_{low^j}^m}{max_m - min_m} \right)$$

Eq. 6.4

Step 5 Calculate summation of d_i^m.

$$CD_i = \sum_{m=1}^{M} d_i^m$$

Eq. 6.5

In Eq. 6.4, $I_{up^i}^m$ is the nearest upper m^{th} objective value for solution i. Meanwhile, $I_{low^j}^m$ represent the nearest lower m^{th} objective value for solution i. In this case, if the objective value is located at the first or last place in the r_m, the max_m and min_m value is used to replace the nearest value respectively.

For *Pbest*, if the current particle has larger *CD* compared with existing *Pbest*, the *Pbest* is replaced with its current position, otherwise, the existing *Pbest* is reused. Meanwhile, *Gbest* is selected as the highest *CD* among all Pareto optimal solutions. In the proposed MODPSO, *Gbest* does not represent the most optimum solution as in traditional PSO, but it will be the leader to update the swarm position and velocity for the next iteration.

6.3.4.4 Update Position and Velocity

The final step in MODPSO is to update swarm position and velocity. The purpose of this step is to establish a new swarm set that follows the current *Pbest* and *Gbest*. In original PSO, the position and velocity are updated using the following formulae:

$$X_i^{t+1} = X_i^t + V_i^{t+1}$$

Eq. 6.6

$$V_i^{t+1}=c_1V_i^t+c_2\left(Pbest_i^t-X_i^t\right)+c_3(Gbest_t-X_i^t)$$

Eq. 6.7

All the operations in Eq. 6.6 and 6.7 can easily be performed for continuous problems. However, for discrete problems, the following discrete position and velocity update procedure was proposed to replace the original operations (Lv and Lu, 2010; Rameshkumar et al., 2005).

Subtraction operator (position – position): This operation is found in Eq. 6.7, $\left(Pbest_i^t-X_i^t\right)and(Gbest_t-X_i^t)$ and produces the velocity. Let $X_1{}^t$= [$x_{1,1}$, $x_{1,2}$, $x_{1,3}$, $x_{1,4}$, $x_{1,5}$, $x_{1,6}$, $x_{1,7}$], X_2^t= [$x_{2,1}$, $x_{2,2}$, $x_{2,3}$, $x_{2,4}$, $x_{2,5}$, $x_{2,6}$, $x_{2,7}$] and V_1^t= X_1^t - X_2^t. In this case, if x_1 and x_2 in the j^{th} position is equal, then the v_1 = 0. Otherwise, v_1 = x_1. For example, $Pbest_1$= [1 4 5 7 2 6 3] and X_1= [6 3 5 7 1 4 2] is considered. The output from this operation by using this rule is V_1 = [1 4 0 0 2 6 3].

Addition operator (position + velocity): For the addition of position and velocity in Eq. 6.6, if the j^{th} element of velocity (v_j) is equal to zero, the j^{th} position value (x_j^t) is inserted into the j^{th} element of new position (x_j^{t+1}). In the meantime, if v_j is nonzero and does not appear in new position, then x_j^{t+1} = v_j. Otherwise, x_j^{t+1} is equal to zero. Later, for x_j^{t+1} is equal to zero, the unassigned value from $X_1{}^t$ is inserted to replace zero value in $X_1{}^{t+1}$. For example, $X_1{}^t$= [6 3 5 7 1 4 2] and V_1 = [1 4 0 0 2 6 3] is considered. The output from this operation is $X_1{}^{t+1}$ = [1 4 5 7 2 6 3].

Multiplication operator (coefficient + velocity): This operation is performed to make an adjustment on the influence of *Pbest* and *Gbest* on the new velocity. This operation can be represented as $V_2 = c \times V_1$, where coefficient $c \in [0, 1]$ is used to control the effect of V_1 that inherit in V_2. For this purpose, a random number, $rand \in [0, 1]$ is generated. If $rand<c$, $v_2 = v_1$, or else, v_2 = 0. In this work, the coefficient c_1, c_2 and c_3 are set at 0.7.

Addition operator (velocity + velocity): This operation is performed to sum up the velocities in Eq. 6.7. For new velocity, $V = V_1 + V_2$, the j^{th} element of V can be derived as follows:

$$v_j = \begin{cases} v_{1,j} \text{ if } v_{1,j} \neq 0,\ v_{2,j} = 0 \\ v_{1,j} \text{ if } v_{1,j} \neq 0,\ v_{2,j} \neq 0,\ r < cp \\ v_{2,j} \text{ otherwise} \end{cases}$$

Eq. 6.8

In Eq. 6.8, r is a random number between 0 and 1, while $cp \in [0, 1]$ is inheriting constant that influences either v_1 or v_2 into new velocity.

Table 6.4 presents the comparison of the proposed MODPSO with NSGA-II, DPSO and MOPSO algorithms in terms of major algorithm stages. In general, the MODPSO algorithm applied similar strategies with NSGA-II for Initialisation, Evaluation and Selection stages, but different in the Regeneration stage. The MODPSO's Regeneration strategy used the discrete position and velocity update procedure adopted from the DPSO algorithm. Meanwhile, in contrast to MOPSO, the proposed MODPSO used different Selection and Regeneration strategies to handle multi-objective problems.

Table 6.4: Comparison of the NSGA-II, DPSO, MOPSO and the proposed MODPSO

Algorithm stage	**NSGA-II**	**DPSO**	**MOPSO**	**MODPSO**
Initialisation	Random initial population	Random initial particles	Random initial particles	Random initial particles
Evaluation	Individual fitness evaluation	Weighted based evaluation	Individual fitness evaluation	Individual fitness evaluation
Selection	Best Crowding Distance of non-dominated solution	Best weighted fitness	Random selection from less density hypercube of non-dominated solution	Best Crowding Distance of non-dominated solution
Regeneration	Crossover and mutation operators	Discrete PSO procedure to update position and velocity	Standard PSO operators to update position and velocity	Discrete PSO procedure to update position and velocity

6.4 Experimental Design

In order to test the proposed MODPSO, the experimental design is set up. The main purpose of this experiment is to test the performance of the proposed MODPSO compared with other algorithms using a set of wide range of problem difficulties. In Chapter 5, a tuneable test problem generator for ASP and ALB has been developed. The results indicate that the ASP and ALB problem difficulties can be increased using a larger number of tasks (*n*), lower Order Strength (*OS*), lower Time Variability Ratio (*TV*) and higher Frequency Ratio (*FR*).

For experimental purposes, each of the input variables is divided into five levels from low to high difficulty as shown in Table 6.5. Then a reference variable setting (datum) is selected as a baseline, while the rest of the problem variable settings are generated by changing only one variable value at a time. In total, there are 17 test problems (including reference setting) generated from one reference variable setting. In order to confirm algorithm performance, three different reference variable setting will be used (Level 1, 3 and 5). Therefore, the complete number of test problems included in this experiment is 51, as shown in Table 6.6. The bold problem settings (Problems 1, 18 and 35) represent the reference variable settings for Level 1, 3 and 5 respectively.

Table 6.5: Level of tuneable input setting

Level	*n*	*OS*	*TV*	*FR*
1	15	0.6	8	0.2
2	20	0.5	6	0.3
3	40	0.4	4	0.4
4	60	0.3	3	0.6
5	80	0.2	2	0.8

Table 6.6: Experimental design for integrated ASP and ALB

Test Problem Variable for Reference Setting at Level 1					Test Problem Variable for Reference Setting at Level 3					Test Problem Variable for Reference Setting at Level 5				
Problem	*n*	*OS*	*TV*	*FR*	Problem	*n*	*OS*	*TV*	*FR*	Problem	*n*	*OS*	*TV*	*FR*
1	**15**	**0.6**	**8**	**0.2**	**18**	**40**	**0.4**	**4**	**0.4**	**35**	**80**	**0.2**	**2**	**0.8**
2	20	0.6	8	0.2	19	15	0.4	4	0.4	36	15	0.2	2	0.8
3	40	0.6	8	0.2	20	20	0.4	4	0.4	37	20	0.2	2	0.8
4	60	0.6	8	0.2	21	60	0.4	4	0.4	38	40	0.2	2	0.8
5	80	0.6	8	0.2	22	80	0.4	4	0.4	39	60	0.2	2	0.8
6	15	0.5	8	0.2	23	40	0.6	4	0.4	40	80	0.6	2	0.8
7	15	0.4	8	0.2	24	40	0.5	4	0.4	41	80	0.5	2	0.8
8	15	0.3	8	0.2	25	40	0.3	4	0.4	42	80	0.4	2	0.8
9	15	0.2	8	0.2	26	40	0.2	4	0.4	43	80	0.3	2	0.8
10	15	0.6	6	0.2	27	40	0.4	8	0.4	44	80	0.2	8	0.8
11	15	0.6	4	0.2	28	40	0.4	6	0.4	45	80	0.2	6	0.8
12	15	0.6	3	0.2	29	40	0.4	3	0.4	46	80	0.2	4	0.8
13	15	0.6	2	0.2	30	40	0.4	2	0.4	47	80	0.2	3	0.8
14	15	0.6	8	0.3	31	40	0.4	4	0.2	48	80	0.2	2	0.2
15	15	0.6	8	0.4	32	40	0.4	4	0.3	49	80	0.2	2	0.3
16	15	0.6	8	0.6	33	40	0.4	4	0.6	50	80	0.2	2	0.4
17	15	0.6	8	0.8	34	40	0.4	4	0.8	51	80	0.2	2	0.6

The MODPSO for integrated ASP and ALB problems has been coded using MATLAB software. For performance comparison purposes, six other algorithms are used.

i. Multi-Objective Genetic Algorithm (MOGA): This algorithm was used in Choi et al. (2009) to optimise ASP problems. It is chosen because genetic algorithm is one of the most frequently used algorithms for solving and optimising ASP problems (Rashid et al., 2012b). In common with this work, it used task-based representation for ASP problems.

ii. Ant Colony Optimisation (ACO): This algorithm has been used for simple assembly line balancing problems in Bautista and Pereira (2007). This algorithm is selected based on its popularity. In addition, ant colony algorithm is also one of the most frequently used algorithms to solve and optimise ALB problems (Rashid et al., 2012b).

iii. Hybrid Genetic Algorithm (HGA): HGA has been proposed by Chen and selected based on citation popularity for integrated ASP and ALB optimisation (Chen et al., 2002).

iv. Elitist Non-Dominated Sorting Genetic Algorithm (NSGA-II): NSGA-II was introduced by Deb (2001). This algorithm is selected because of its popularity in multi-objective optimisation.

v. Multi-Objective Particle Swarm Optimisation (MOPSO): The MOPSO acronym was introduced by Coello and Lechuga to extend the PSO application for Pareto-based multi-objective optimisation instead of weighted-based approach in earlier versions (Coello Coello and Lechuga, 2002).

vi. Discrete Particle Swarm Optimisation (DPSO): DPSO proposed by Rameshkumar for discrete problems. Instead of using normal mathematical operations to update position and velocity in PSO, this algorithm introduced special procedures to incorporate the discrete problem (Rameshkumar et al., 2005).

In this work, the population or swarm size is set at 20 with 500 iterations. For each problem, 30 simulation runs with different random seeds are performed and the output from each run are gathered and filtered to attain the non-dominated solution.

To evaluate the performance of each algorithm when dealing with different complexity problems, the following performance indicators, adopted from Deb (2001) and Yoosefelahi et al. (2012), are used. The details of these indicators have been explained in Section 5.4.2.

i. Number of non-dominated solution in Pareto optimal ($\tilde{n}$)

ii. Error Ratio (*ER*)

iii. Generational Distance (*GD*)

iv. *Spacing*

v. Maximum Spread ($Spread_{max}$)

6.5 Experimental Results

Figure 6.5 shows the number of non-dominated solutions in Pareto optimal ($\tilde{n}$) for all test problems using different algorithms. This figure shows that the proposed MODPSO performed better than other algorithms in all test problems. In the majority of test problems, there are significant gaps between MODPSO and other algorithms in terms of $\tilde{n}$ found. According to the output pattern, larger problem size will arise with broader gaps between MODPSO and other algorithms.

The Error Ratio (*ER*) for all algorithms is presented in Figure 6.6. From 51 test problems, MODPSO algorithms performed better in 82% of the problems. While the remaining 18% of problems are led by NSGA-II, where most of these problems involve larger task numbers (60 and 80 tasks). However, the mean of *ER* using MODPSO for all test problems remains the smallest (0.34) compared to NSGA-II (0.57) and other algorithms (between 0.81-0.93).

Meanwhile, Figure 6.7 presents the Generational Distance (*GD*) for algorithms throughout the test problems. For this indicator, MODPSO also performed better in 82% of the problems in almost similar problems as in *ER*. This is because the *GD* is measured between the solutions with the nearest Pareto solution. When the number of non-Pareto solutions is increased (higher *ER*), the average distance from Pareto solution will also generally increase.

Figure 6.8 shows the performance of *Spacing* indicator that leads for different algorithms. For this indicator, MOPSO algorithm had performed better in 37% of test problems. Then follow MODPSO (22%), HGA (18%), DPSO (15%), MOGA (6%) and ACO (2%).

For Maximum Spread ($Spread_{max}$) in Figure 6.9, all algorithms show almost similar graph patterns with small gaps between each other. For this indicator, the MODPSO algorithm performed better in 71% of test problems. In this case, MODPSO achieved better performance in the problems with larger numbers of tasks, as it performed better in all test problems with 60 and 80 tasks.

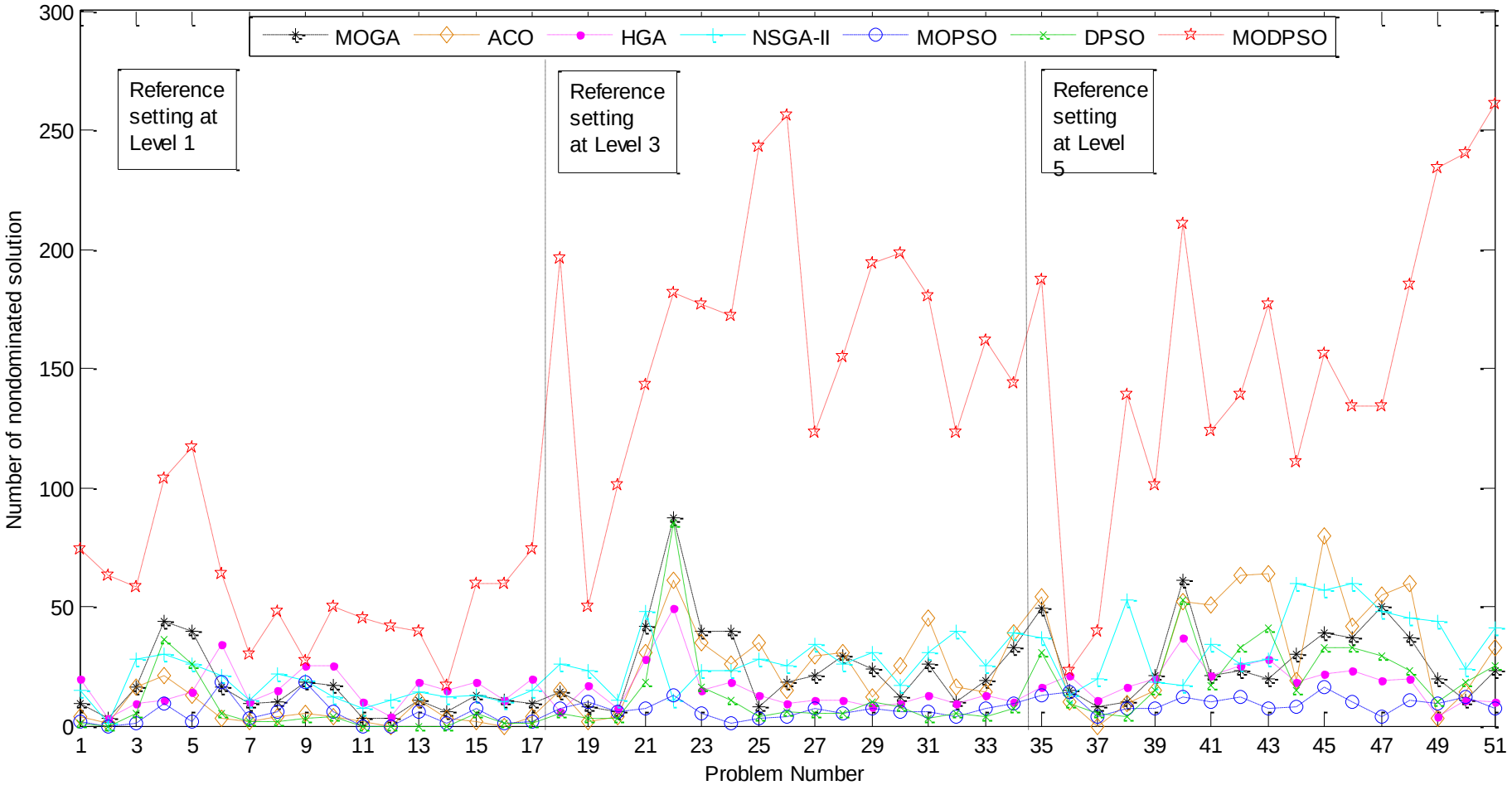


Figure 6.5: Number of non-dominated solution in Pareto optimal throughout test problems

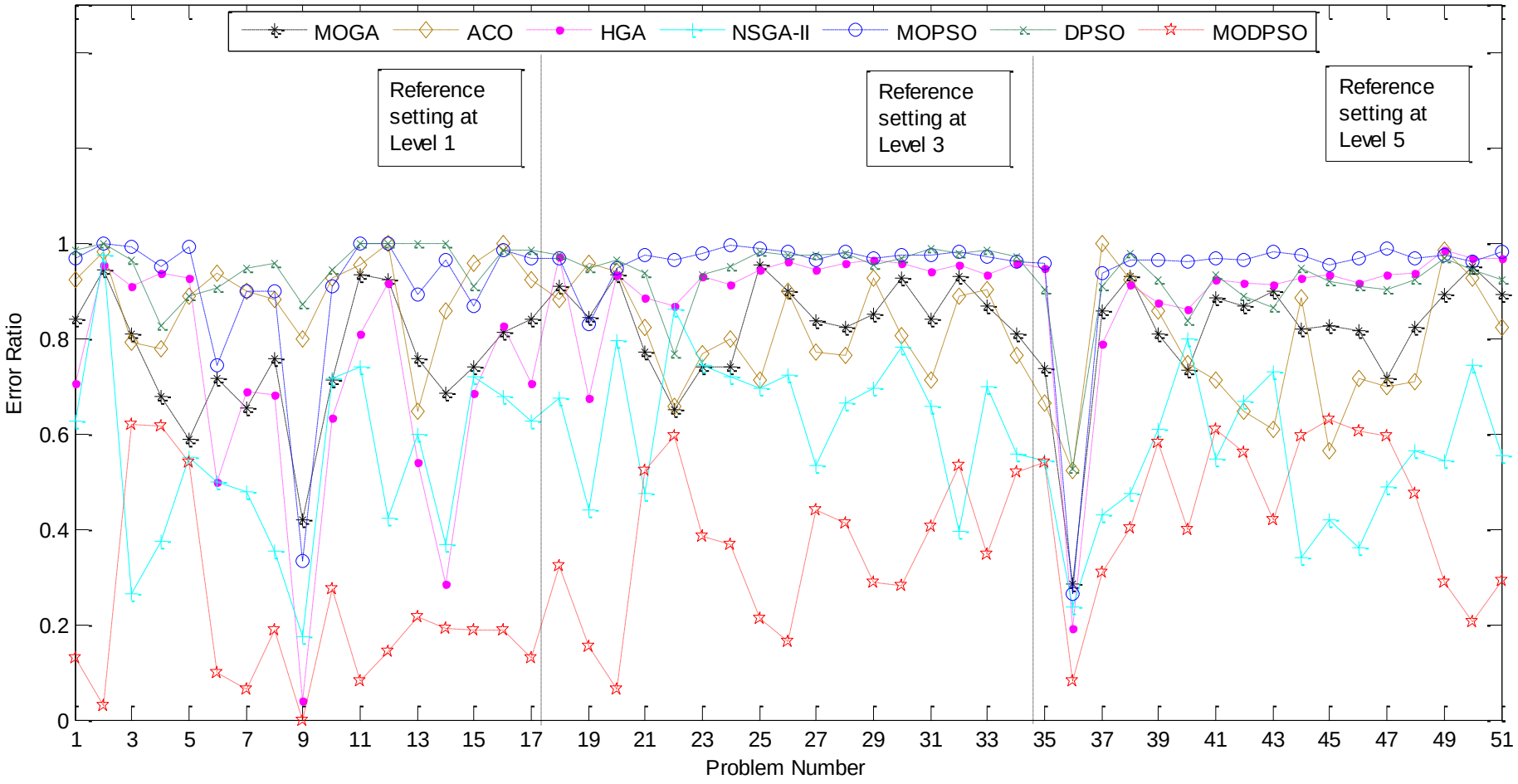


Figure 6.6: Error Ratio throughout test problems

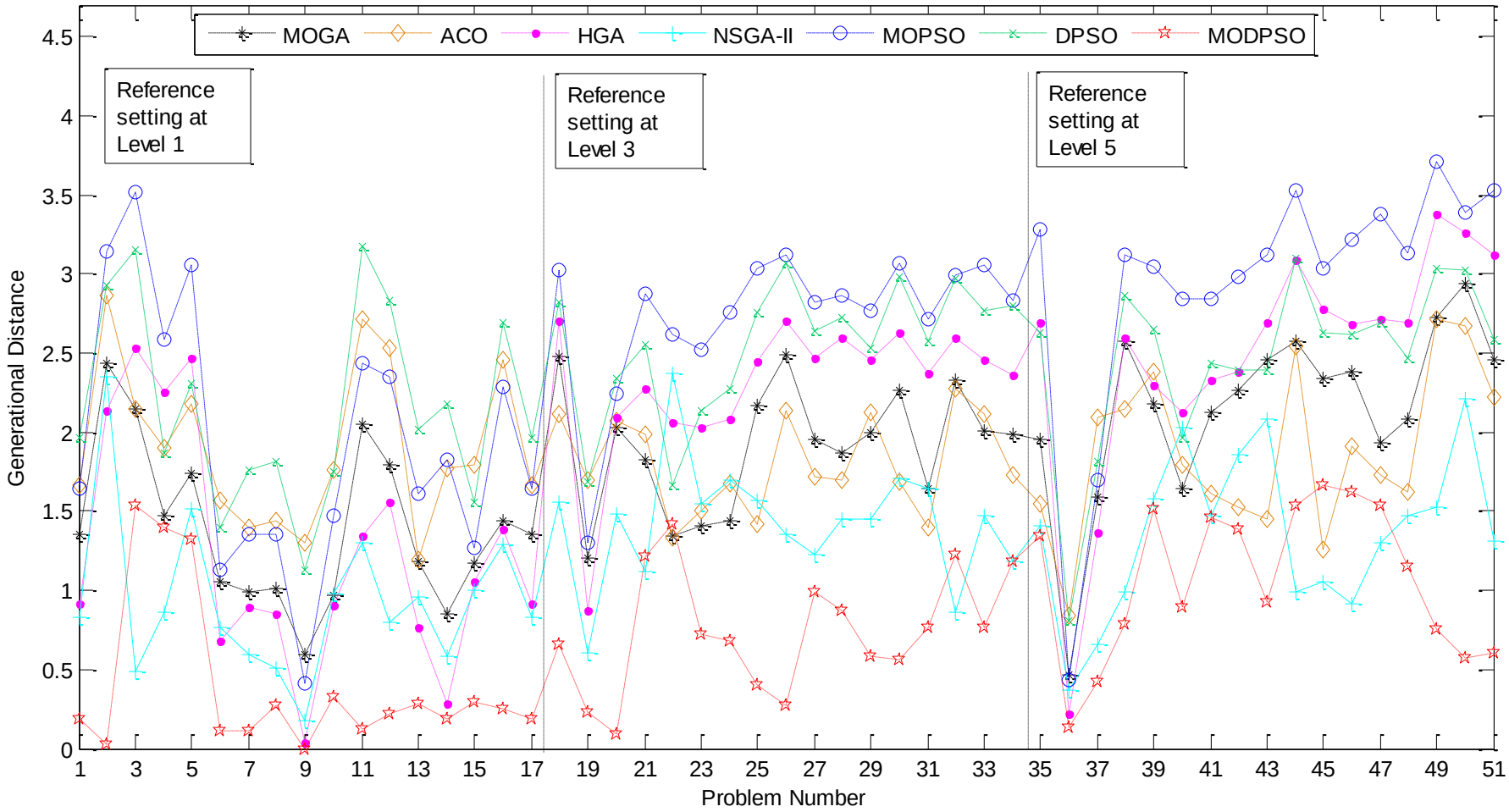


Figure 6.7: Generational Distance throughout test problems

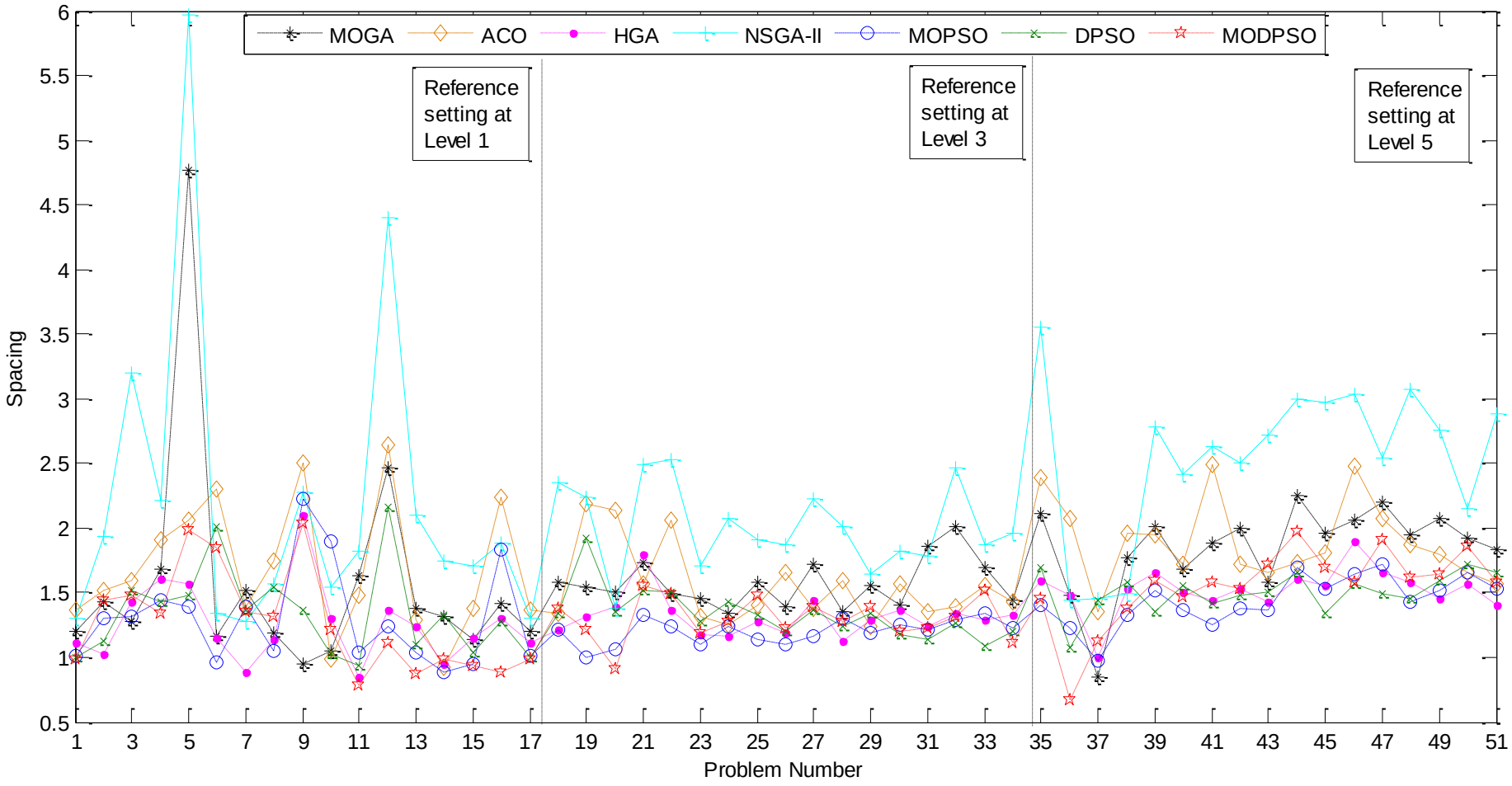


Figure 6.8: Solution *Spacing* throughout test problems

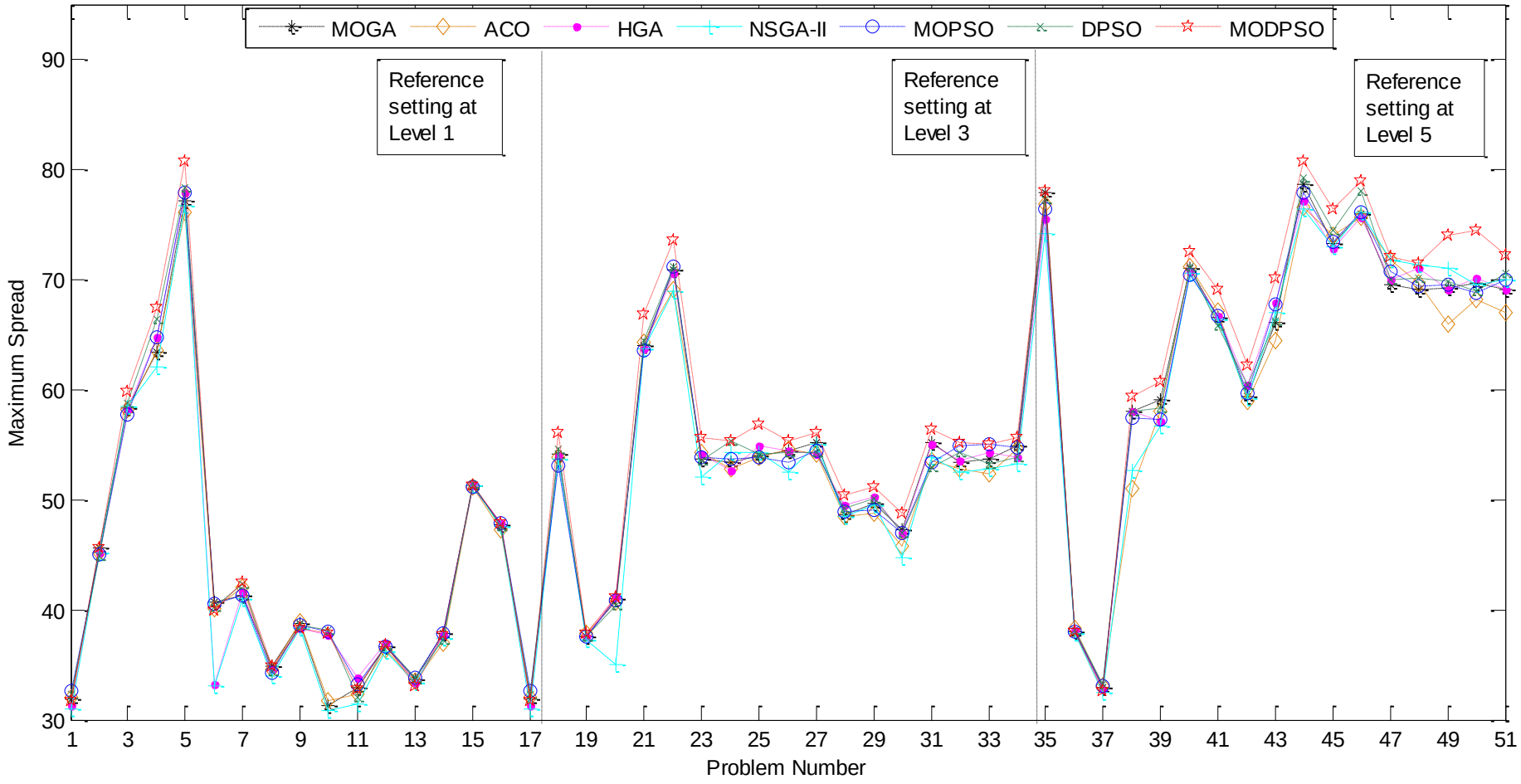


Figure 6.9: Maximum spread throughout test problems

6.6 Discussion of MODPSO Algorithm

Table 6.7 presents the mean of performance indicators that obtained using different reference variable settings. The number in brackets represents the algorithm ranking based on the mean of each indicator.

Table 6.7: Mean of performance indicators by different reference setting
*(Number in bracket shows algorithm ranking based on mean indicator value)

Ref. setting	Indicator	Algorithms						
		MOGA	ACO	HGA	NSGA-II	MOPSO	DPSO	MODPSO
Level 1	$\tilde{\eta}$	14.0000 (4)	5.5882 (5)	15.4118 (3)	15.7059 (2)	4.8235 (7)	5.3529 (6)	57.2353 (1)
	ER	0.7530 (4)	0.8906 (5)	0.6901 (3)	0.5386 (2)	0.9034 (6)	0.9504 (7)	0.2174 (1)
	GD	1.3890 (4)	1.9020 (5)	1.2328 (3)	0.9307 (2)	1.9463 (6)	2.1462 (7)	0.4029 (1)
	Spacing	1.5743 (5)	1.6869 (6)	1.2509 (1)	2.2123 (7)	1.2932 (3)	1.3366 (4)	1.2690 (2)
	Spread-max	43.2394 (4)	43.0415 (6)	43.2337 (5)	42.2759 (7)	43.7500 (3)	43.8658 (2)	44.1101 (1)
Level 3	$\tilde{\eta}$	25.7059 (3)	25.5882 (4)	14.4706 (5)	27.1176 (2)	6.2941 (7)	11.6471 (6)	164.6471 (1)
	ER	0.8419 (4)	0.8210 (3)	0.9222 (5)	0.6531 (2)	0.9647 (7)	0.9544 (6)	0.3533 (1)
	GD	1.9089 (4)	1.8061 (3)	2.3055 (5)	1.4281 (2)	2.7429 (7)	2.5469 (6)	0.7438 (1)
	Spacing	1.5697 (6)	1.5558 (5)	1.3120 (3)	2.0204 (7)	1.2011 (1)	1.3345 (4)	1.3033 (2)
	Spread-max	52.9868 (2)	52.4293 (6)	52.9695 (4)	51.8761 (7)	52.8396 (5)	52.9752 (3)	54.5288 (1)
Level 5	$\tilde{\eta}$	27.9412 (4)	36.7059 (3)	18.9412 (6)	36.7647 (2)	9.5882 (7)	23.1765 (5)	152.7059 (1)
	ER	0.8076 (4)	0.7638 (3)	0.8756 (5)	0.5318 (2)	0.9247 (7)	0.8939 (6)	0.4460 (1)
	GD	2.1575 (4)	1.8858 (3)	2.4961 (6)	1.3660 (2)	2.9583 (7)	2.4755 (5)	1.0768 (1)
	Spacing	1.8601 (5)	1.8992 (6)	1.5246 (3)	2.5535 (7)	1.4429 (1)	1.5062 (2)	1.5530 (4)
	Spread-max	64.8871 (3)	63.9614 (7)	64.8369 (5)	64.4443 (6)	64.8487 (4)	65.2652 (2)	67.2345 (1)

According to this table, the MODPSO algorithm consistently performed better in $\tilde{n}$, *ER*, *GD* and $Spread_{max}$ for all reference settings. Meanwhile, for the *Spacing* indicator, HGA and MOPSO algorithms have shown better performance than MODPSO. Based on this table, the proposed MODPSO algorithm has better performance in all indicators except *Spacing* indicator.

In the *Spacing* indicator, all non-dominated solutions found by particular algorithms are taken into account, regardless of Pareto or non-Pareto solutions. This indicator shows the uniformity of the space between one solution and the nearest solution. Thus, the algorithm which generated more non-dominated solutions has a greater chance of produce better (smaller) *Spacing* within a similar distribution area.

The solution distribution is also important in achieving better *Spacing*. This is because the solutions only distributed to a particular side/s of solution space will have better *Spacing* compared to solutions distributed uniformly through an entire solution space, even though the number of non-dominated solutions is much smaller. As an example in problem 3, the number of non-dominated solutions found using MODPSO and MOGA are 152 and 84 solutions respectively, but MOGA came out with better *Spacing* compared to MODPSO. Figure 6.10 shows the distribution of solutions for problem 3, considering direction change and tool change objectives using both algorithms. From this figure, the MODPSO solution is distributed in larger solution space with a larger number of solutions, but it has worse *Spacing* compared to MOGA.

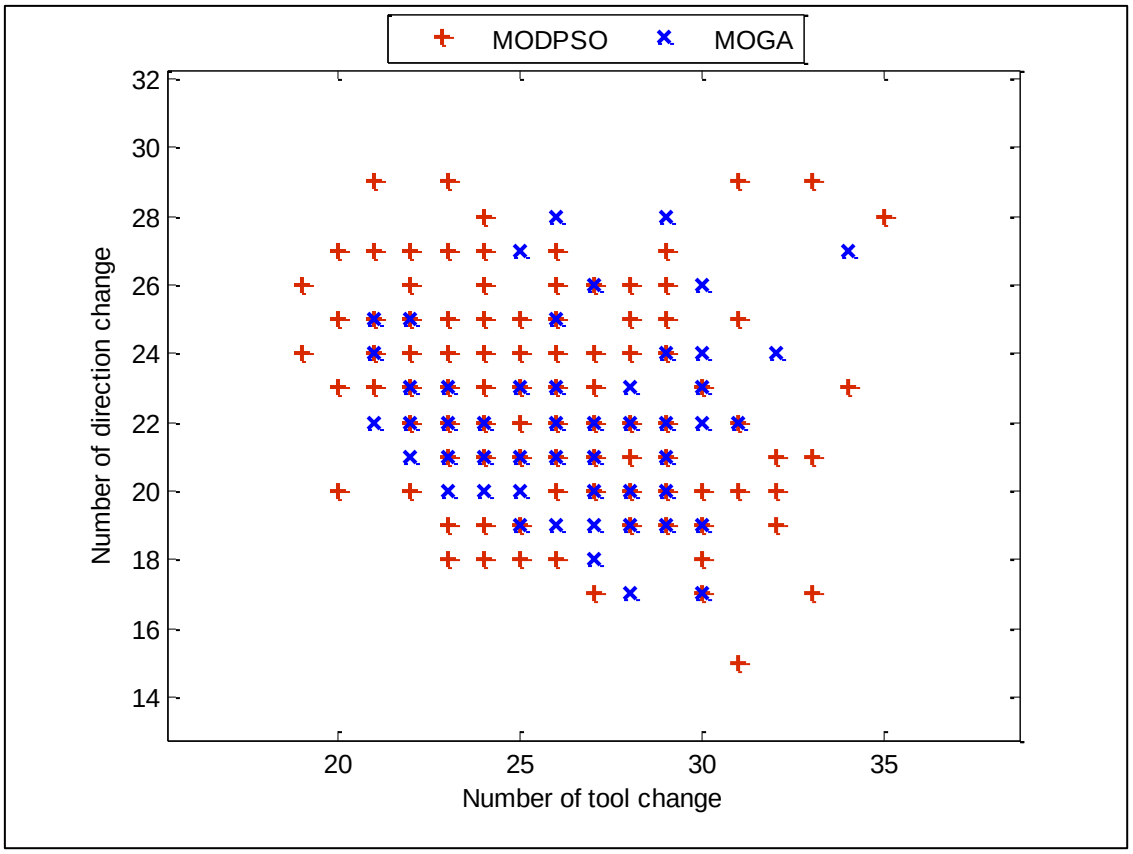


Figure 6.10: Scatter plot for problem 3 using MOGA and MODPSO algorithms

Based on the means of performance indicators, the algorithms with the basis of Genetic Algorithm (GA) show good performance behind the proposed MODPSO. The NSGA-II consistently shows good performance in three indicators behind MODPSO although it does not perform as well in *Spacing* and $Spread_{max}$ indicators. Meanwhile the MOGA algorithm shows medium performance in most indicators for all reference setting. By calculated mean, this algorithm is located between the third and fourth rank. However, HGA shows inconsistent performance from one reference setting level to another. For reference setting at Level 1, HGA shows quite good performance at third ranking. But when the reference setting changed to Levels 3 and 5, the HGA mean ranking dropped to fourth and fifth position respectively.

On the other hand, ACO algorithm shows improvements from one reference setting level to another level for $\tilde{\eta}$, *ER* and *GD*. On average, the ACO placed in the fifth rank when compared with all algorithms. The remaining two algorithms,

DPSO and MOPSO are placed in the sixth and seventh rank according to indicator means. Both algorithms did not perform well in $\tilde{\eta}$, *ER* and *GD* but shows good performance in *Spacing* and $Spread_{max}$ indicators.

The performance of DPSO shows that this algorithm, which was designed with the weighted objective function approach, is unsuitable for finding non-dominated solutions, although it used an efficient Regeneration procedure as in MODPSO. Meanwhile the MOPSO's performance shows that the original PSO operator to update position and velocity is not good enough for discrete problems. On the other hand, the NSGA-II that performed efficiently in three indicators shows that the Selection strategy based on Crowding Distance of the non-dominated solution work effectively, since the MODPSO that adopted a similar strategy also did well. Based on the performance of NSGA-II and DPSO algorithms, the proposed MODPSO algorithm has inherited good features from NSGA-II and DPSO because the MODPSO algorithm mainly adopted strategies from these algorithms.

The results in Table 6.7 also indicate that the proposed MODPSO consistently performed better than GA-based algorithms (i.e. MOGA, HGA and NSGA-II) for all indicators except *Spacing* in all reference settings. It shows that the MODPSO is able to optimise integrated ASP and ALB problems from various difficulty levels efficiently, compared to GA-based algorithms.

6.6.1 Statistical Tests

To test the significance of the results, statistical tests have been performed. In this case, the ANOVA test was carried out to test for any significant improvements between results obtained by one algorithm compared to other algorithms. The null hypothesis stated that there is no significant difference among all algorithms' means. When the null hypothesis is accepted, it means that there is no significant improvement achieved by any algorithms. The summary of the ANOVA test is presented in Table 6.8.

Table 6.8: Summary of ANOVA test

	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
SSB	512147.60	14.30	121.93	35.03	183.90
SSW	301524.1	8.1442	137.805	74.207	72260.1
MSB	85358	2.38395	20.3224	5.83856	30.648
MSW	861.5	0.02327	0.3937	0.21202	206.457
*f**	3.68	3.68	3.68	3.68	3.68
f	99.08	102.45	51.62	27.54	0.15

SSB: Sum of square between groups

SSW: Sum of square within groups

MSB: Mean squares between groups

MSW: mean squares within groups

*f**: critical f-value

f: calculated f-value

In order to accept the null hypothesis, the calculated *f*-value must be smaller than critical *f*-value (*f**). The *f** that is obtained from the *f*-distribution table at the 0.05 confidence interval is 3.86 (Coolidge, 2000). Based on Table 6.8, only the *f*-value for $Spread_{max}$ fulfils the requirement to accept the null hypothesis. Meanwhile, the *f*-values for $\tilde{\eta}$, *ER*, *GD* and *Spacing* indicators show larger values compared to *f**. It means that four out of five performance indicators rejected the null hypothesis, which means that there are significant differences between algorithms. In this case, it shows that there are significant improvements achieved by at least one algorithm compared to others. Meanwhile, the acceptance of null hypothesis by the $Spread_{max}$ indicator shows that all algorithms are able to explore the extreme minimum and maximum values in the search space.

However the ANOVA test did not tell us the exact algorithms that have significant means difference. Therefore, a *posteriori* test known as Tukey's Honestly Significant Different test (Tukey's HSD test) is performed to identify any significant improvement achieved by the proposed MODPSO compared to other algorithms. The Tukey's HSD test is only conducted for the performance

indicators that rejected the null hypothesis (*ñ*, *ER*, *GD* and *Spacing*) since only these groups show the significant difference between algorithms.

The summary of Tukey's HSD test is presented in Table 6.9.

Table 6.9: Summary of Tukey's HSD test

		Absolute Mean Difference Between MODPSO and Algorithm			
Indicator (HSD*)		*ñ* (11.102099)	*ER* (0.089074)	*GD* (0.366382)	*Spacing* (0.268868)
Algorithm	MOGA	**102.313725**	**0.461927**	**1.077286**	**0.292912**
	ACO	**102.235294**	**0.486218**	**1.123465**	**0.338841**
	HGA	**108.588235**	**0.490418**	**1.270296**	0.012637
	NSGA-II	**98.3333333**	**0.235592**	**0.500439**	**0.886947**
	MOPSO	**117.960784**	**0.592025**	**1.807975**	0.062702
	DPSO	**111.470588**	**0.593996**	**1.648349**	0.017292

Table 6.9 presents the absolute mean difference between MODPSO and other algorithms. The number in brackets shows the critical HSD value (HSD*) that was calculated based on Tukey's table (Coolidge, 2000). The HSD* value for algorithm *i* is calculated as follows.

$$HSD_i^* = q.\sqrt{\frac{MSW_i}{n}}$$

Eq. 6.9

The *q* value is acquired from Tukey's table. MSW is the mean squares within groups from ANOVA test, and *n* is the number of data in each group. When the absolute mean difference between MODPSO and a particular algorithm is larger than HSD*, it means that the significant improvement has been identified between these two algorithms. Based on Table 6.9, the significant

improvement has been achieved by the proposed MODPSO compared to all other algorithms for $\tilde{\eta}$, *ER* and *GD* indicators. In the meantime, the significant *Spacing* improvements are observed between MODPSO and MOGA, ACO and NSGA-II, but not with HGA, MOPSO and DPSO. This result is consistent with earlier findings in Figure 6.8 and Table 6.7 which prioritise the HGA, MOPSO and DPSO algorithms together with MODPSO for the *Spacing* indicator.

The Tukey's HSD test result explained that the proposed MODPSO performed well to converge to Pareto optimal solutions since the indicators directly linked with it (i.e. $\tilde{\eta}$, *ER* and *GD*) show significant improvement compared to other algorithms. On the other hand, the MODPSO only shows significant improvement in some cases in terms of uniformity of the found solution. Meanwhile, no significant improvement is found for the $Spread_{max}$ although a small difference, as presented in Figure 6.9, is noticed.

6.7 Chapter Summary

In this chapter, a Multi-Objective Discrete Particle Swarm Optimisation (MODPSO) algorithm has been proposed to optimise integrated ASP and ALB problems. In contrast with existing algorithms, MODPSO used the Pareto-based approach to deal with multi-objective problems, and it adopted discrete procedures instead of standard mathematical operators to update its position and velocity. A set of 51 test problems with a different range of difficulties has been used to test the performance of MODPSO compare with other algorithms.

The results show that the MODPSO performed better in all test problems in finding non-dominated solution ($\tilde{\eta}$), 82% of the problems in Error ratio (*ER*), 82% of the problems in Generational Distance (*GD*), 22% of the problems in *Spacing* and 71% of the problems in maximum spread ($Spread_{max}$). Meanwhile, the results shown in Table 6.7 indicate that the MODPSO performed better in four out of five performance indicators in all difficulty levels. This result shows that the proposed MODPSO has successfully overcome the under-performance of GA-based algorithms for test problems with a larger number of tasks.

Statistical tests have been conducted to identify any significant improvement achieved by the proposed MODPSO. Statistical tests concluded that the MODPSO has shown significant improvements compared to other algorithms in converging to Pareto optimal solutions. In terms of solution uniformity, the significant improvement achieved by MODPSO is only applied to certain comparison algorithms. Furthermore, no significant improvement is achieved for the $Spread_{max}$ using the MODPSO. Therefore, it can be concluded that the proposed MODPSO has shown good performance in terms of solution quality towards Pareto optimal solutions. However, the proposed MODPSO has limited performance in term of solution uniformity as attained by the *Spacing* indicator.

In summary, this chapter has achieved the following goals:

- The detail of proposed MODPSO algorithm has been presented.
- The experimental design to test the MODPSO algorithm using a different range of problem difficulties has been explained.
- The experimental results of MODPSO algorithm, including the statistical test have been presented and discussed.

CHAPTER 7

OPTIMISATION OF INTEGRATED MIXED-MODEL ASP AND ALB

The existing optimisation works regarding integrated ASP and ALB are limited to simple assembly lines which only run one homogeneous product on a serial line layout. This chapter aims to extend the application of the MODPSO algorithm proposed in Chapter 6 to optimise more general types of assembly problems, that is, mixed-model ASP and ALB. The mixed-model assembly line runs different product models in arbitrarily inter-mixed sequence on a single assembly line (Scholl and Becker, 2006).

This chapter aims to achieve the following goals:

- Explain the background and significance of integrated mixed-model ASP and ALB.
- Present the formulation of integrated mixed-model ASP and ALB.
- Explain the experimental design for integrated mixed-model ASP and ALB using the MODPSO algorithm.
- Present and discuss the experimental results of integrated mixed-model ASP and ALB.

7.1 Background and Significance

In general, assembly line problems in the literature are classified into two categories, i.e. simple and generalised assembly line problems (Scholl and Becker, 2006). The simple assembly line only runs one homogenous product, on a serial line layout, and all workstations are equally equipped with machines and workers (Scholl and Becker, 2006). The generalised assembly line includes all problems that are not simple assembly problems, such as U-shape, two-sided lines, mixed-model and multi-model assembly line (Tasan and Tunali, 2008). Up to Chapter 6, we have only considered the ASP and ALB problem within simple assembly line category.

The mixed-model assembly line is widely used in various industries to produce a variety of products on one single assembly line (Zhu et al., 2012b). This type of assembly line is important in industry because sharing the different model of products in the same assembly line can save investment cost (Hu et al., 2008). In addition to that, the mixed-model assembly line can also absorb the fluctuation of demand for different models using an assembly line (Hu et al., 2008). Therefore, by integrating the ASP and ALB optimisation for mixed-model assembly, the benefits from integrated optimisation and mixed-model assembly as presented earlier, will be obtained.

The integrated mixed-model ASP and ALB problem is more challenging compared to mixed-model ALB and integrated ASP and ALB for single-model. ASP and ALB problems are individually categorised as NP-hard combinatorial problems, where the solution space is excessively increased when the number of tasks is increased (Goldwasser and Motwani, 1997; Wee and Magazine, 1982). When the optimisation of both activities is performed together, the problem difficulties will be increased since all the related factors such as geometric information, assembly tool and time are considered concurrently in this stage (Tseng et al., 2008). Furthermore, with mixed-model assembly problems it is more difficult to achieve optimum solution for all models compared to simple assembly problems (Becker and Scholl, 2006). Therefore,

the formulated problem of integrated mixed-model ASP and ALB will be more challenging to solve and to optimise when compared to individual optimisation of mixed-model ASP or ALB and also integrated ASP and ALB for simple assembly.

7.2 Integrated Mixed-Model ASP and ALB Formulation

The mixed-model assembly line runs one product, but with different models. For example, in vehicle production, although the assembly line runs a specific car type, it also produces different model variants, such as right or left hand drive and manual or automatic transmission. Other than that, some of the cars require additional accessories to fulfil specific customer requirements. In this case, the assembly line only runs similar products, i.e. specific car types, but the assembly process will be different due to model differences. To formulate the mixed-model assembly problem, the following assumptions are applied:

i. All workstations are equally equipped with tools, machines and/or workers.
ii. Similar assembly tasks should be assigned into same workstation.
iii. The assembly data is deterministic for all assembly tasks.
iv. No contrary precedence constraint among all models.

The most common approach to express the mixed-model assembly problem is by transforming the precedence graphs into a joint graph, as used in many existing mixed-model Assembly Line Balancing works (Kara et al., 2011; Tambe, 2006). By using this approach, similar optimisation procedure, as used in single-model (i.e. Chapter 6), can be implemented with small changes in evaluation steps. The objective functions for integrated mixed-model ASP and ALB are presented in the following section.

The joint graph represents the precedence constraint for all models. For example, an assembly line runs two models of a product, Model A and Model B. The precedence graphs for both models are shown in Figure 7.1(a) and (b).

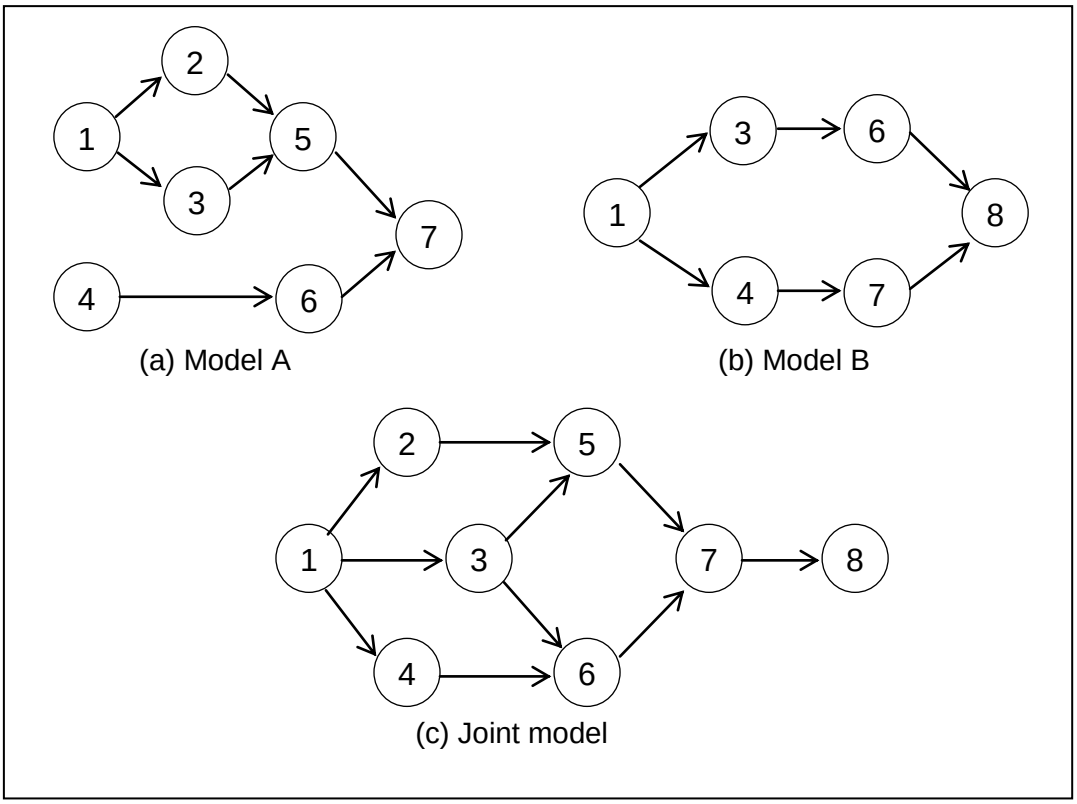


Figure 7.1: Precedence graph of (a) Model A, (b) Model B and (c) Joint Model

To establish the joint graph, the followers for specific tasks in each model are bundled together in one graph. For example in Figure 7.1, the followers for task 1 in Model A are tasks 2 and 3, while tasks 3 and 4 in Model B. The combinations of task 1 followers from both models are tasks 2, 3 and 4, as shown in the joint graph. The joint graph is updated by removing the shortest repetitive routes from the graph. In the example above, the route connecting tasks 4 and 7 in Model B is removed from the Joint Model because task 7 cannot be started although task 4 has been performed, because there is dependence on the completion of task 6 in Model B. Once the joint graph has been established, similar representation schemes as in simple assembly line problems can be used, except for assembly data representation.

In a mixed-model assembly line, the assembly data set should represent data for each model. In this case, the assembly data for similar tasks within different

models might be different, depending on the actual processing task. For example, the data for Model A and B in Figure 7.1 is presented in Table 7.1. In Table 7.1, entries marked '0' shows that the particular task is inapplicable to that model. For example, task 8 is inapplicable for Model A, while tasks 2 and 5 are inapplicable for Model B.

Table 7.1: Assembly data for Model A and Model B

	Model A			Model B		
Task	*D*	*T*	*M*	*D*	*T*	*M*
1	+*y*	1	21	+*y*	4	18
2	-*x*	1	9	0	0	0
3	+*y*	2	14	+*z*	2	25
4	+*z*	3	11	+*z*	3	11
5	+*z*	1	28	0	0	0
6	+*x*	3	17	+*y*	4	16
7	+*y*	2	12	-*x*	5	12
8	0	0	0	+*z*	5	21

D: Direction *T*: Tool *M*: Time

7.2.1 Objective Function

To evaluate the fitness, in mixed-model assembly problem, the mean of fitness value from *G* different models is used. For the g^{th} models and n^{th} task in i^{th} workstation;

Objective 1: Minimise the mean of total direction changes

$$\overline{n}_{dc} = \frac{1}{G}\left(\sum_{g=1}^{G}\sum_{i=1}^{nws}\sum_{n=2}^{Ni} DC_n\right)$$

Eq. 7.1

Where *Ni* is the total number of assembly task in workstation *i*.

$$DC_n = \begin{cases} 0, & \text{if the assembly direction of } n^{th} \text{ task is similar to the previous task} \\ 1, & \text{if the assembly direction of } n^{th} \text{ task is not the same as the previous task} \end{cases}$$

Objective 2: Minimise the mean of total tool changes

$$\overline{n}_{tc} = \frac{1}{G}\left(\sum_{g=1}^{G}\sum_{i=1}^{nws}\sum_{n=2}^{Ni} TC_n\right)$$

Eq. 7.2

$$TC_n = \begin{cases} 0, & \text{if the assembly tool of } n^{th} \text{ task is similar to the previous task} \\ 1, & \text{if the assembly tool of } n^{th} \text{ task is not the same as the previous task} \end{cases}$$

Objective 3: Minimise the mean of cycle times

$$\overline{ct} = \frac{1}{G}\sum_{g=1}^{G} ct_g; \qquad ct_g: \text{Cycle time for } g^{th} \text{ model}$$

Eq. 7.3

Objective 4: Minimise number of workstations

Number of workstations (*nws*) is determined once the assembly tasks assignments are completed. The number of workstations generated for all models will be the same because similar tasks within different model are assigned into similar workstations.

Objective 5: Minimise the mean of workload variations

$$\overline{v} = \frac{1}{G}\sum_{g=1}^{G} \frac{\sum_{i=1}^{nws}(ct_g - pt_i^g)}{nws}; \qquad pt_i^g: \text{processing time in } i^{th} \text{workstation for model } g$$
$$nws: \text{total number of workstation}$$

Eq. 7.4

7.2.2 Numerical Example

Consider the assembly sequence $fseq_1$ = [1 4 3 6 2 5 7 8] generated from the joint graph in Figure 7.1. Figure 7.2 shows the assembly task assignment for $fseq_1$ = [1 4 3 6 2 5 7 8] given the following maximum allowable cycle time: for Model A, ct_{maxA} = 52 and for Model B, ct_{maxB} = 46 time unit. In the first workstation, the total processing time for Model B when assigning tasks 1, 4 and 3 is equal to 54, which exceeds ct_{maxB}. Therefore, the assembly task 3 for both models is assigned to the second workstation.

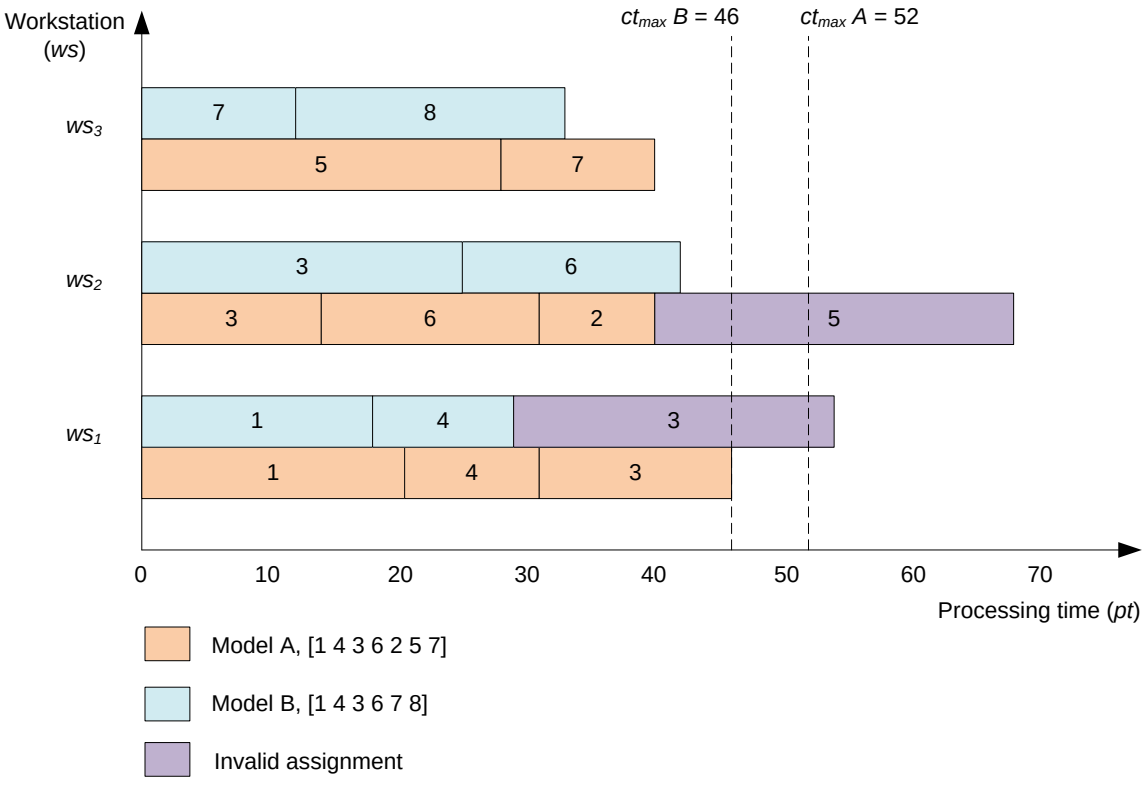


Figure 7.2: Example of the assignment of assembly tasks into workstations

The direction and tool change for different models are calculated within all workstations, and then the mean from all models is determined.

Table 7.2: Example of assembly direction and tool changes calculation

Workstation		ws_1		ws_2			ws_3			
$fseq_1$		1	4	3	6	2	5	7	8	
Time	Model A	21	11	14	17	9	28	12	0	
	Model B	18	11	25	16	0	0	12	21	
Dir.										n_{dc}
	Model A	$+x$	$+z$	$+y$	$+x$	$-x$	$+z$	$+y$	0	4
	Model B	$+y$	$+z$	$+z$	$+y$	0	0	$-x$	$+z$	3
Tool										n_{tc}
	Model A	1	3	2	3	1	1	2	0	4
	Model B	4	3	2	4	0	0	5	5	2

From Table 7.2, the Objective 1 and 2 can be calculated as follows:

$$\overline{n}_{dc}=\frac{4+3}{2}=3.5$$

$$\overline{n}_{tc}=\frac{4+2}{2}=3$$

According to this assignment procedure, the cycle time for Model A (ct_A) is 40 as found in ws_2 and ws_3, while ct_B = 41 time unit (in ws_2). Therefore Objective 3 can be calculated as follows;

$$\overline{ct}=\frac{40+41}{2}=40.5 \text{ time unit}$$

Objective 4 (to minimise number of workstations) can be determined once the task assignment into workstations is completed. For this example, the number of workstations is;

nws = 3 workstations

Based on the assignment procedure above, the total number of workstations is three workstations. Therefore, the Objective 5 mean of workload variation is as follows.

$$\overline{v}=\frac{1}{2}\left[\frac{(40\text{-}32)+(40\text{-}40)+(40\text{-}40)}{3}+\frac{(41\text{-}29)+(41\text{-}41)+(41\text{-}33)}{3}\right]$$

= 4.667 time unit per workstation.

7.3 Experimental Design

The main purpose of this experiment is to test the performance of the MODPSO compared to other algorithms in a wide range of problem difficulties. In Chapter 5, a tuneable test problem generator for ASP and ALB has been developed. The results indicate that the ASP and ALB problem difficulties can be increased using larger numbers of tasks (*n*), lower Order Strength (*OS*), lower Time Variability ratio (*TV*) and higher Frequency Ratio (*FR*). For the testing of integrated mixed-model ASP and ALB, the precedence graph that is generated in the tuneable test problem generator is assumed as joint model. Then, a small modification to assembly data generation is made by increasing the number of generated assembly data sets to three sets instead of only one set in the original test problem generator. In this case, the different assembly data set represents different product models.

For experimental purposes, each of the input variables is divided into five levels from low to high difficulty values, as shown in Table 7.3. Then a reference variable setting (datum) is selected as a baseline, while the rest of the problem variable settings are generated by changing only one variable value at a time. In total, there are 17 test problems (including reference setting) generated from one reference variable setting. In order to confirm algorithm performance, three different reference variable settings will be used (Levels 1, 3 and 5). Therefore, the complete number of test problems in this experiment is 51, as shown in

Table 7.4. The bold problem setting (Problems 1, 18 and 35) represents the reference variable settings for Level 1s, 3 and 5 respectively.

Table 7.3: Level of tuneable input setting

Level	*n*	*OS*	*TV*	*FR*
1	15	0.6	8	0.2
2	20	0.5	6	0.3
3	40	0.4	4	0.4
4	60	0.3	3	0.6
5	80	0.2	2	0.8

Table 7.4: Experimental design for mixed-model ASP and ALB

Test Problem Variable for Reference Setting at Level 1					Test Problem Variable for Reference Setting at Level 3					Test Problem Variable for Reference Setting at Level 5				
Problem	*n*	*OS*	*TV*	*FR*	Problem	*n*	*OS*	*TV*	*FR*	Problem	*n*	*OS*	*TV*	*FR*
1	**15**	**0.6**	**8**	**0.2**	**18**	**40**	**0.4**	**4**	**0.4**	**35**	**80**	**0.2**	**2**	**0.8**
2	20	0.6	8	0.2	19	15	0.4	4	0.4	36	15	0.2	2	0.8
3	40	0.6	8	0.2	20	20	0.4	4	0.4	37	20	0.2	2	0.8
4	60	0.6	8	0.2	21	60	0.4	4	0.4	38	40	0.2	2	0.8
5	80	0.6	8	0.2	22	80	0.4	4	0.4	39	60	0.2	2	0.8
6	15	0.5	8	0.2	23	40	0.6	4	0.4	40	80	0.6	2	0.8
7	15	0.4	8	0.2	24	40	0.5	4	0.4	41	80	0.5	2	0.8
8	15	0.3	8	0.2	25	40	0.3	4	0.4	42	80	0.4	2	0.8
9	15	0.2	8	0.2	26	40	0.2	4	0.4	43	80	0.3	2	0.8
10	15	0.6	6	0.2	27	40	0.4	8	0.4	44	80	0.2	8	0.8
11	15	0.6	4	0.2	28	40	0.4	6	0.4	45	80	0.2	6	0.8
12	15	0.6	3	0.2	29	40	0.4	3	0.4	46	80	0.2	4	0.8
13	15	0.6	2	0.2	30	40	0.4	2	0.4	47	80	0.2	3	0.8
14	15	0.6	8	0.3	31	40	0.4	4	0.2	48	80	0.2	2	0.2
15	15	0.6	8	0.4	32	40	0.4	4	0.3	49	80	0.2	2	0.3
16	15	0.6	8	0.6	33	40	0.4	4	0.6	50	80	0.2	2	0.4
17	15	0.6	8	0.8	34	40	0.4	4	0.8	51	80	0.2	2	0.6

The MODPSO for integrated mixed-model ASP and ALB problem has been coded using MATLAB software. As used in Chapter 6, six other algorithms that have been used to optimise integrated ASP and ALB are implemented, as follows:

i. Multi-Objective Genetic Algorithm (MOGA) (Choi et al., 2009).

ii. Ant Colony Optimisation (ACO) (Bautista and Pereira, 2007).

iii. Hybrid Genetic Algorithm (HGA) (Chen et al., 2002).

iv. Elitist Non-Dominated Sorting Genetic Algorithm (NSGA-II) (Deb K., 2001).

v. Multi-Objective Particle Swarm Optimisation (MOPSO) (Coello Coello and Lechuga, 2002).

vi. Discrete Particle Swarm Optimisation (DPSO) (Rameshkumar et al., 2005).

The reason behind the selection of these algorithms has been explained in Section 6.4. In this work, the population or swarm size is set at 20 with 500 iterations. For each problem, 30 simulation runs with different random seeds are performed and the output from each run are collected and filtered to find the non-dominated solution set.

To measure the performance of the algorithms in optimising integrated mixed-model ASP and ALB, the following performance indicators are used. The explanation of these indicators has been presented in Section 5.4.2.

i. Number of non-dominated solution in Pareto optimal, $\tilde{\eta}$

ii. Error Ratio, *ER*

iii. Generational Distance, *GD*

iv. *Spacing*

v. Maximum Spread, $Spread_{max}$

7.4 Optimisation Results

Figure 7.3 to Figure 7.7 inclusive present the performance indicators resulting from experiments. For the non-dominated solution in Pareto optimal ($\tilde{\eta}$) indicator (Figure 7.3), the MODPSO comes out with better solution sets in 96% of test problems, while the remaining 4% belong to NSGA-II. The Error Ratio (*ER*) indicator, as presented in Figure 7.4, also shows that the leading algorithms are MODPSO and NSGA-II. The MODPSO and NSGA-II show better performance in 41.5% and 58.5% of cases respectively. Both algorithms also dominate the best performance for the Generational Distance (*GD*) indicator with 43% better performance for MODPSO and 53% for NSGA-II, shown in Figure 7.5. Meanwhile, the *Spacing* indicator (Figure 7.6) shows a different pattern, where the largest percentages of better performance are MOPSO (22), followed by HGA (20%), ACO (19%), DPSO (17%), MOGA (14%), MODPSO (6%) and NSGA-II (2%). On the other hand, the *Maximum Spread* indicator (Figure 7.7) that measure the extent of solution distribution presents that the MODPSO algorithm produce better solutions in 70% of the problems. The MOPSO perform better in 18%, while the remaining balances are shared amongst DPSO (6%), MOGA (4%) and HGA (2%).

Table 7.5 presents the mean of performance indicators for all test problems. Based on the mean values, the best performance of $\tilde{\eta}$ indicator is observed in MODPSO and followed by NSGA-II algorithms. The best mean performance for *ER* and *GD* indicators is achieved by NSGA-II, while the MODPSO is in second place. Two PSO-based algorithms, MOPSO and DPSO lead the mean of *Spacing* indicator. Furthermore, the PSO-based algorithms also show better performance compared to other algorithms in the $Spread_{max}$ indicator.

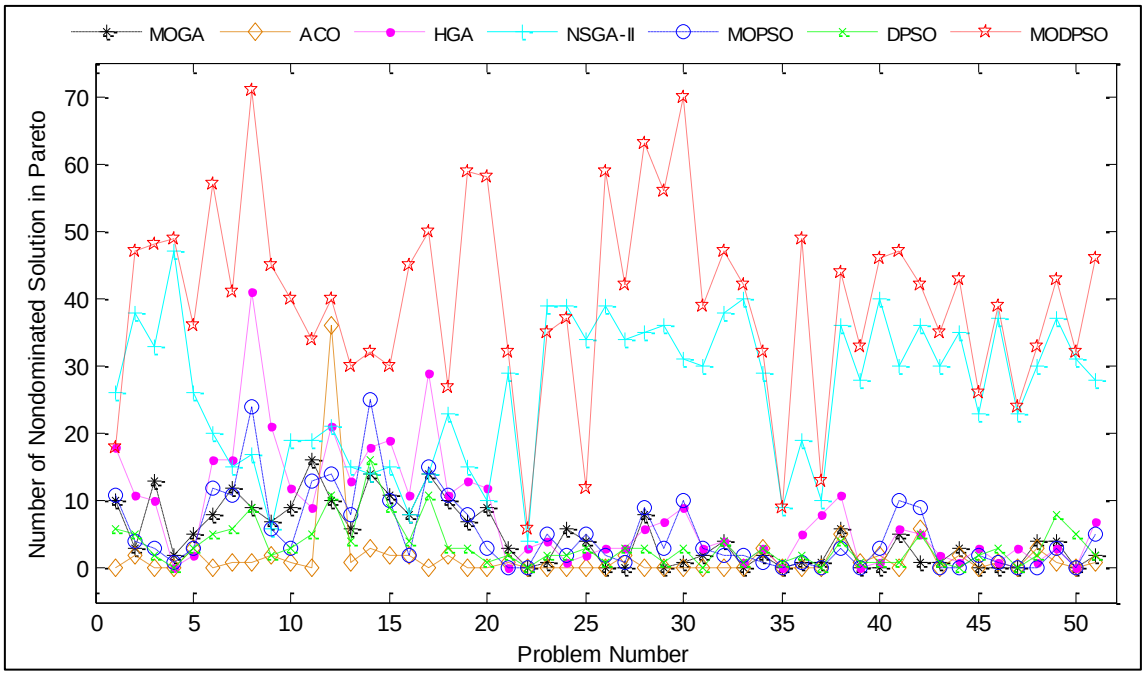


Figure 7.3: Number of non-dominated solution in Pareto optimal

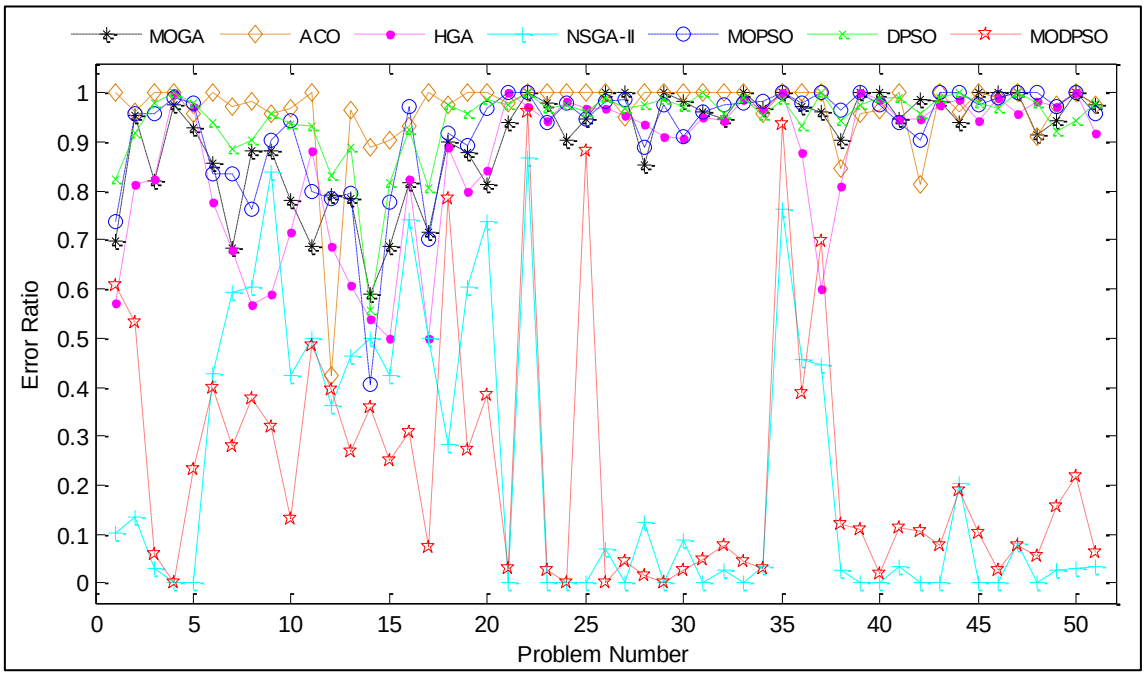


Figure 7.4: Plot of Error Ratio throughout test problems

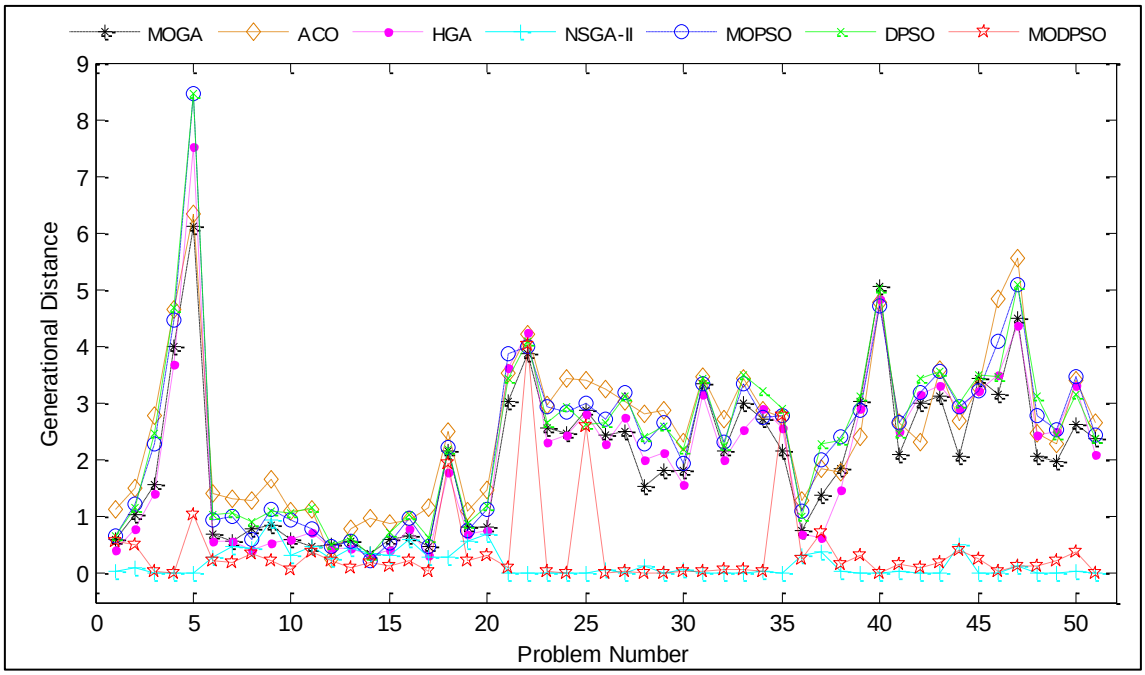


Figure 7.5: Plot of Generational Distance throughout test problems

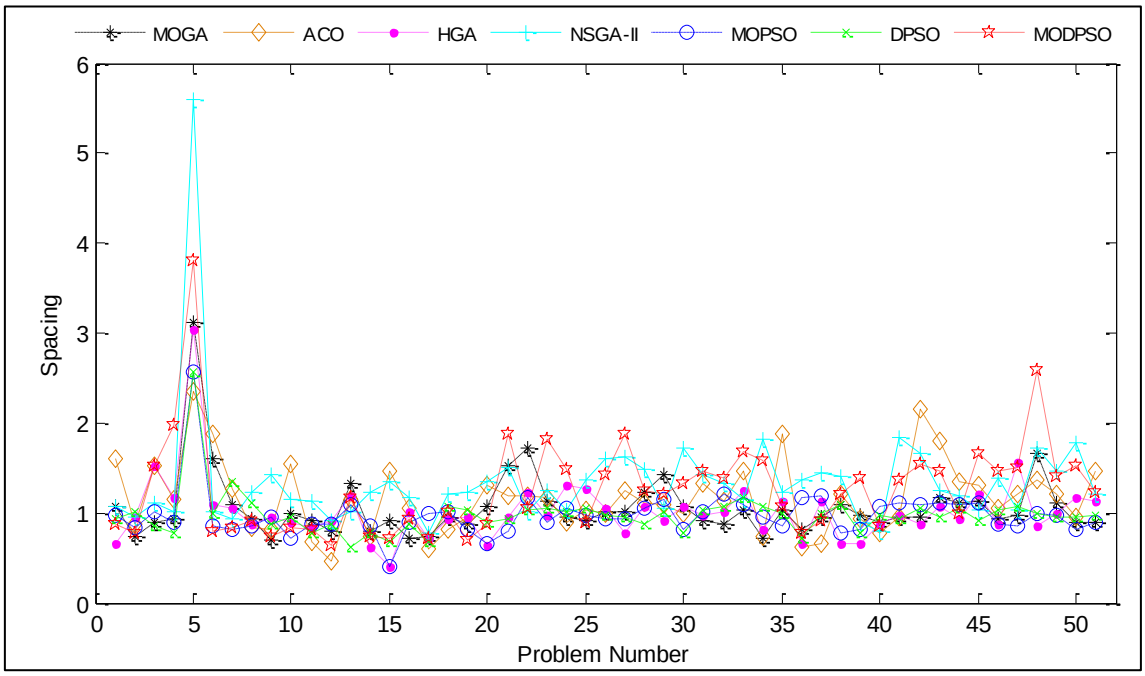


Figure 7.6: Plot of Spacing throughout test problems

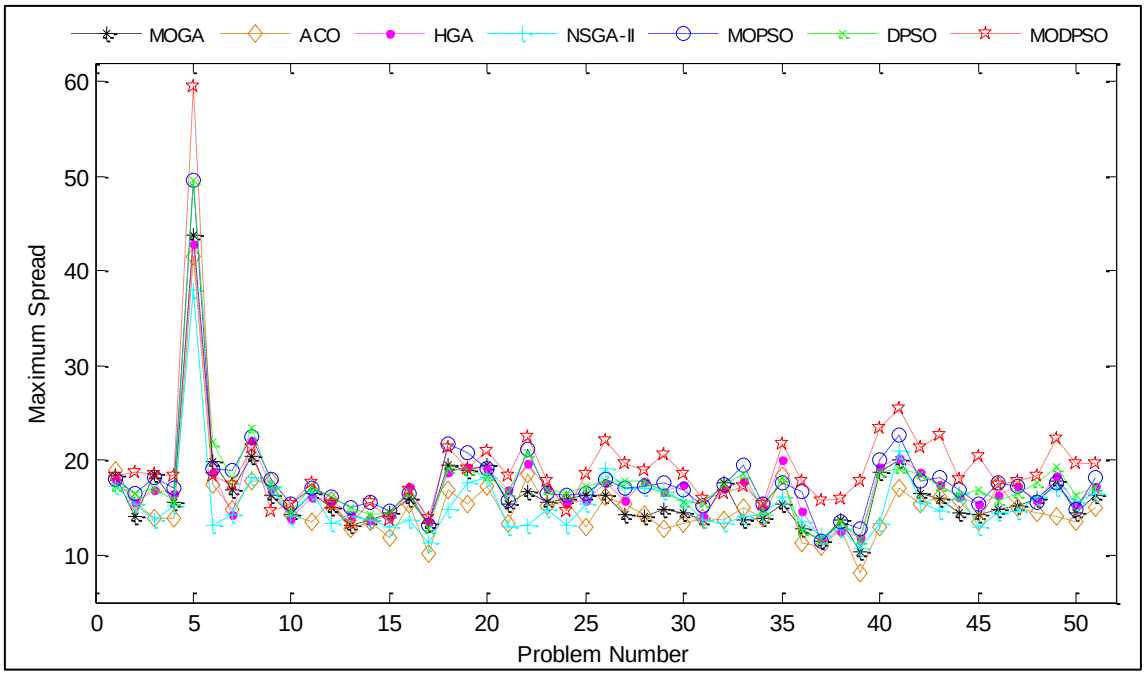


Figure 7.7: Plot of Maximum Spread throughout test problems

Table 7.5: Mean of performance indicators

Indicator	Algorithm						
	MOGA	ACO	HGA	NSGA-II	MOPSO	DPSO	MODPSO
$\tilde{\eta}$[1]	4.7843	1.9020	8.0588	27.2353	5.2745	3.4902	41.0196
ER[2]	0.9037	0.9632	0.8592	0.1952	0.9230	0.9444	0.2046
GD[2]	1.9951	2.4650	2.0017	0.1753	2.3219	2.3682	0.2696
$Spacing$[2]	1.0281	1.1410	0.9819	1.2898	0.9479	0.9537	1.2318
$Spread_{max}$[1]	15.7278	14.6364	16.5250	14.9729	17.1868	16.8720	18.4656

[1] Larger is better [2] Smaller is better

7.5 Discussion of Integrated Mixed-Model ASP and ALB

In general, the results from the experiments show the performance of algorithms in optimising integrated mixed-model ASP and ALB appear to be dominated by NSGA-II and proposed MODPSO algorithms, especially in four performance indicators (i.e. $\tilde{\eta}$, *ER*, *GD* and $Spread_{max}$). However, further analyses are required to quantify the results. Therefore, a statistical test is conducted to measure the significance of the improvements achieved by the MODPSO in optimising integrated mixed-model ASP and ALB.

The Analysis of Variance (ANOVA) test is carried out to evaluate any significant improvement between the results obtained by different algorithms. The 'null hypothesis' stated that there is no significant improvement among the means of all algorithm results. The alternative hypothesis states that there is significant improvement among the means in the result of at least one algorithm. The null hypothesis will be accepted when the calculated *f*-value is smaller than critical *f*-value (*f**) as suggested in the *f*-distribution table (Coolidge, 2000). The result of the ANOVA test is presented in Table 7.6.

Table 7.6: Summary of ANOVA test

	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
SSB	68996	37.9748	303.6630	5.9401	545.6308
SSW	21629	8.4491	385.9791	27.0017	1826.1
MSB	11499	6.3291	50.6105	0.9900	90.9385
MSW	61.7972	0.0241	1.1028	0.0771	5.2173
*f**	3.6900	3.6900	3.6900	3.6900	3.6900
f	186.081	262.1808	45.8928	12.8327	17.4301

SSB: Sum of square between groups

SSW: Sum of square within groups

MSB: Mean squares between groups

MSW: Mean squares within groups

*f**: critical f-value

f: calculated f-value

The result shows that the calculated *f*-value for all performance indicators is consistently larger than *f** at 0.05 confidence intervals. Therefore, the null hypothesis is rejected and the alternative is accepted for all indicators, which indicates that there are significant improvements achieved for all indicators in at least one algorithm. However, the ANOVA test cannot differentiate the exact improvement of one algorithm in comparison with another algorithm.

Therefore a *posteriori* test known as Tukey's Honestly Significant Difference (HSD) is performed. This test is performed by calculating the absolute mean difference between the results of one algorithm over another algorithm, which is then compared to the critical HSD (HSD*) value. The HSD* value for algorithm *i* is calculated by using Eq. 6.9.

When the absolute mean difference is larger than HSD*, a significant improvement has been identified in one algorithm over another algorithm. At this point, we are interested to know the performance of MODPSO over the other algorithms. Table 7.7 presents the HSD* and absolute mean difference between MODPSO and the other algorithms.

In Table 7.7, the values that are labelled '1' show the MODPSO has a better mean difference over the comparison algorithm, while the values labelled '2' mean that the comparison algorithm has a better mean difference over MODPSO. On the other hand, the bold values in Table 7.7 indicate the significant improvements achieved by MODPSO over other algorithms.

Table 7.7: Summary of Tukey's HSD test for MODPSO algorithm

Indicator (HSD*)		Absolute Mean Difference Between MODPSO and Comparison Algorithm				
		ñ (4.5902)	*ER* (0.0906)	*GD* (0.6131)	*Spacing* (0.1621)	$Spread_{max}$ (1.3337)
Comparison Algorithm	MOGA	**36.2353**[1]	**0.6991**[1]	**1.7255**[1]	0.2037[2]	**2.7379**[1]
	ACO	**39.1176**[1]	**0.7586**[1]	**2.1954**[1]	0.0908[2]	**3.8292**[1]
	HGA	**32.9608**[1]	**0.6547**[1]	**1.7321**[1]	0.2499[2]	**1.9406**[1]
	NSGA-II	**13.7843**[1]	0.0094[2]	0.0944[2]	0.0580[1]	**3.4928**[1]
	MOPSO	**35.7451**[1]	**0.7184**[1]	**2.0523**[1]	0.2839[2]	1.2789[1]
	DPSO	**37.5294**[1]	**0.7399**[1]	**2.0985**[1]	0.2781[2]	**1.5936**[1]

[1]Better absolute mean difference for MODPSO
[2]Better absolute mean difference for comparison algorithm

Based on Table 7.7, the MODPSO algorithm shows better performance and significant improvement when compared to the set of algorithms for *ñ* indicator. The MODPSO also shows significant improvements for *ER* and *GD* indicators compared to other algorithms, with the exception of NSGA-II. In both indicators, the NSGA-II algorithm shows better mean difference compared to MODPSO. However, the differences are insignificant because the absolute mean differences are smaller than HSD*.

Meanwhile, the *Spacing* indicator did not show any significant improvement of MODPSO although it has a better mean difference when compared to NSGA-II. Except for NSGA-II, all other algorithms show better performance over MODPSO, where significant improvements are presented by four algorithms (MOGA, HGA, MOPSO and DPSO). For the $Spread_{max}$ indicator, the MODPSO algorithm shows significant improvement compared to other algorithms, except for MOPSO. In comparison with MOPSO, although no statistically significant improvement is achieved, the MODPSO algorithm still produces a better solution.

In this work, the solution quality towards Pareto optimal is measured using three performance indicators i.e. *ñ*, *ER* and *GD*. The *Spacing* indicator measures the uniformity of the found solutions and $Spread_{max}$ measures the ability of the

algorithm to explore the extreme solutions within the solution space. The results from statistical tests indicate that the MODPSO algorithm shows significant improvement in terms of finding better solutions towards Pareto optimal over comparison algorithms, with the exception of NSGA-II at 0.05 confidence intervals.

Furthermore, the $Spread_{max}$ result means that the MODPSO algorithm is significantly able to explore better extreme solutions when compared to MOGA, ACO, HGA, DPSO and NSGA-II. Meanwhile, in terms of uniformity of solution spread, the MODPSO algorithm did not perform significantly better than other algorithms. The *Spacing* indicator considers all non-dominated solutions found by a particular algorithm, regardless of Pareto or non-Pareto optimal solutions. In general, for similar search space, the algorithm that generated more non-dominated solutions has greater chances to produce better *Spacing*. From the experiment, the mean number of non-dominated solutions generated by the algorithms (regardless of Pareto or non-Pareto optimal), in ascending order, are: NSGA (33.84), ACO (46.9), MODPSO (55.37), MOGA (56.14), HGA (68.91), DPSO (80.47) and MOPSO (85.02). These numbers clearly show that the algorithms which show significant improvement over MODPSO for *Spacing* indicator are the algorithms with larger mean of generated solutions.

The results from experiments and statistical tests summarise that the MODPSO has shown significant improvement over the majority of comparison algorithms in $\tilde{\eta}$, *ER*, *GD* and $Spread_{max}$ indicators. In comparison with all other algorithms, the performance of MODPSO is closely followed by NSGA-II, where the MODPSO only shows significant improvement over NSGA-II in $\tilde{\eta}$ and $Spread_{max}$ indicators. In order to gain a better understanding of the performance difference between MODPSO and NSGA-II, the absolute mean difference between NSGA-II and other algorithms are calculated and presented in Table 7.8.

Table 7.8: Summary of Tukey's HSD test for NSGA-II

		Absolute Mean Difference Between NSGA-II and Comparison Algorithm				
Indicator (HSD*)		$\tilde{\eta}$ (4.5902)	*ER* (0.0906)	*GD* (0.6131)	*Spacing* (0.1621)	$Spread_{max}$ (1.3337)
Comparison Algorithm	MOGA	**22.4510**[1]	**0.7085**[1]	**1.8199**[1]	0.2618[2]	0.7549[2]
	ACO	**25.3333**[1]	**0.7680**[1]	**2.2897**[1]	0.1489[2]	0.3365[1]
	HGA	**19.1765**[1]	**0.6640**[1]	**1.8264**[1]	0.3079[2]	1.5522[2]
	MOPSO	**21.9608**[1]	**0.7278**[1]	**2.1466**[1]	0.3419[2]	2.2139[2]
	DPSO	**23.7451**[1]	**0.7492**[1]	**2.1929**[1]	0.3361[2]	1.8991[2]
	MODPSO	13.7843[2]	0.0094[1]	0.0944[1]	0.0580[2]	3.4928[2]

[1]Better absolute mean difference for NSGA-II
[2]Better absolute mean difference for comparison algorithm

Table 7.8 indicates that the NSGA-II has significant improvements in solution quality leading to Pareto optimal compared to other algorithms except for the MODPSO. In addition, the NSGA-II did not show any significant improvement for solution uniformity (*Spacing*) and extreme solution exploration ($Spread_{max}$). Based on the significant improvement achieved by MODPSO (Table 7.7) and NSGA-II (Table 7.8) over other algorithms, the MODPSO is found to perform better than NSGA-II. This is because the MODPSO has shown significant improvement over NSGA-II in two indicators (i.e. $\tilde{\eta}$ and $Spread_{max}$), whilst there is no significant improvement of NSGA-II over MODPSO algorithms. Furthermore, for the $Spread_{max}$ indicator, the NSGA-II did not show any significant improvement such as MODPSO shows when compared to all other algorithms.

The result from Tukey's HSD test for integrated mixed-model ASP and ALB clearly shows that the MODPSO performed better than other algorithms for all test problems. Another question that arises concerns the problem categories in which the MODPSO algorithm performed best and worst. Therefore, the Tukey's HSD test based on different problem reference settings is conducted. The result of Tukey's HSD test for different problem settings is presented in Table 7.9.

Table 7.9: Summary of Tukey's HSD test for MODPSO by reference setting level

Reference Setting	Algorithm	Absolute Mean Difference Between MODPSO and Algorithm				
		ñ	*ER*	*GD*	*Spacing*	$Spread_{max}$
Level 1	HSD*	9.3491	0.1684	0.9264	0.3109	2.3693
	MOGA	**32.3529**[1]	**0.5009**[1]	0.7618[1]	0.0116[2]	0.8080[1]
	ACO	**37.8235**[1]	**0.6412**[1]	**1.2778**[1]	0.1152[1]	2.1079[1]
	HGA	**25.8824**[1]	**0.4142**[1]	0.6812[1]	0.0322[2]	0.8376[1]
	NSGA	**20.8235**[1]	0.0966[1]	0.0903[1]	0.1924[1]	2.2812[1]
	MOPSO	**31.9412**[1]	**0.5373**[1]	**0.9917**[1]	0.0924[2]	0.1576[2]
	DPSO	**35.7059**[1]	**0.5927**[1]	**1.1062**[1]	0.0839[2]	0.0122[1]
Level 3	HSD*	7.2889	0.1436	0.8001	0.2325	1.9374
	MOGA	**40.5882**[1]	**0.7850**[1]	**2.0228**[1]	0.2749[2]	**2.7749**[1]
	ACO	**43.2941**[1]	**0.8292**[1]	**2.5720**[1]	0.2616[2]	**3.7624**[1]
	HGA	**38.8824**[1]	**0.7747**[1]	**2.0196**[1]	0.3462[2]	1.5431[1]
	NSGA	**13.0588**[1]	0.0448[2]	0.2236[2]	0.0036[1]	**3.3176**[1]
	MOPSO	**40.0000**[1]	**0.7975**[1]	**2.3401**[1]	0.3913[2]	0.8924[1]
	DPSO	**41.7059**[1]	**0.8142**[1]	**2.3461**[1]	0.3576[2]	1.2606[1]
Level 5	HSD*	5.4125	0.1002	0.9376	0.2957	2.6973
	MOGA	**35.7647**[1]	**0.8115**[1]	**2.3920**[1]	0.3247[2]	**4.6306**[1]
	ACO	**36.2353**[1]	**0.8054**[1]	**2.7364**[1]	0.1261[2]	**5.6173**[1]
	HGA	**34.1176**[1]	**0.7751**[1]	**2.4955**[1]	0.3712[2]	**3.4412**[1]
	NSGA	**7.4706**[1]	0.0799[2]	0.1497[2]	0.0219[2]	**4.8795**[1]
	MOPSO	**35.2941**[1]	**0.8204**[1]	**2.8249**[1]	0.3679[2]	**3.1018**[1]
	DPSO	**35.1765**[1]	**0.8127**[1]	**2.8433**[1]	0.3927[2]	**3.5081**[1]

[1]Better absolute mean difference for MODPSO
[2]Better absolute mean difference for comparison algorithm

Based on Table 7.9, the MODPSO shows significant improvement in *ñ* indicator over all algorithms for all reference settings. For the *ER* indicator, the MODPSO consistently demonstrates significant improvement over other algorithms except for NSGA-II. Meanwhile for the *GD* indicator in low level reference setting (Level 1), significant improvements for MODPSO are only found over ACO, MOPSO and DPSO algorithms. However, when the reference setting is changed to

medium (Level 3) and high (Level 5) levels, significant improvements are also observed in comparison with MOGA and NSGA-II.

On the other hand, the MODPSO consistently failed to show any significant improvement over any algorithm for *Spacing* indicator. For the $Spread_{max}$ indicator, the proposed algorithm also failed to show significant improvements in low level reference setting. However, when the reference setting is moved to medium level, the MODPSO shows significant improvement over MOGA, ACO and NSGA-II. Finally, in the problem with high level reference setting, significant improvements are achieved by MODPSO over all other algorithms. From this result, the best performance of MODPSO is found in the problem with high reference setting. Meanwhile, the weakest performance is in the problems with low level reference setting, even though the overall performance in this problem category is still better than other algorithms.

The superior performance of MODPSO in optimising integrated mixed-model ASP and ALB arises because this algorithm was specifically developed for discrete multi-objective optimisation problems. This algorithm uses similar procedure for Initialisation, Evaluation and Selection strategies as in NSGA-II. The NSGA-II is another algorithm specifically developed for multi-objective optimisation problems that also performed well in this application. The proposed MODPSO inherits the good features from NSGA-II, which results in good performance in $\tilde{\eta}$, *ER* and *GD*. At the same time, the MODPSO algorithm retains better search space exploration as achieved by PSO-based algorithms (Table 7.5) in $Spread_{max}$ indicator. However, the MODPSO algorithm did not maintain the good performance of *Spacing* indicator as obtained by the MOPSO and DPSO algorithms.

7.6 Chapter Summary

This chapter formulates and studies the optimisation of integrated mixed-model Assembly Sequence Planning (ASP) and Assembly Line Balancing (ALB) problems. The Multi-Objective Discrete Particle Swarm Optimisation (MODPSO), previously developed for single-model assembly optimisation is extended to optimise integrated mixed-model ASP and ALB. A set of test problems with a different range of difficulties has been used to test the performance of MODPSO in optimising integrated mixed-model ASP and ALB.

The experimental results indicate that, in general, the MODPSO algorithm performed better than other comparable algorithms. Statistical tests concluded that the MODPSO has shown significant improvement in converging to Pareto optimal solutions and exploring the extreme solutions in search space. The statistical tests also concluded that the MODPSO performed best in the problem with high level of difficulty. Meanwhile, the weakest performance is in the problem at low difficulty level, although it still performed better than the compared algorithms.

The work in this chapter not only initiates the effort on integrated mixed-model ASP and ALB optimisation, but also indicates that the MODPSO algorithm is able to optimise this problem better than the compared algorithms. One downside of MODPSO is its incapability of generating uniformly spaced solutions, as presented by *Spacing* indicator.

In summary, this chapter has achieved the following goals:

- The background and significance of integrated mixed-model ASP and ALB have been explained.
- The formulation of integrated mixed-model ASP and ALB has been presented.
- The experimental design for integrated mixed-model ASP and ALB using the MODPSO algorithm has been explained.

- ❖ The experimental results of integrated mixed-model ASP and ALB has been presented and discussed.

CHAPTER 8

VALIDATION

This chapter presents the validation of the proposed MODPSO algorithm. In Chapter 6, an algorithm called Multi-Objective Discrete Particle Swarm Optimisation (MODPSO) was proposed to optimise integrated single-model ASP and ALB problems. The MODPSO was extensively tested using a wide range of problem difficulties generated from the test problem generator. The results so far show that the MODPSO algorithm performed significantly better than comparable algorithms in terms of solution quality towards the Pareto optimal solution.

Besides the integrated single-model ASP and ALB optimisation, the author also formulated and initiated the integrated mixed-model ASP and ALB optimisation in Chapter 7. This type of problem is important to enhance product variations without investing in different assembly lines (Hu et al., 2008). In addition, the mixed-model assembly line can also absorb the fluctuation of demand of different models using an assembly line (Hu et al., 2008). Extensive testing using a wide range of problem difficulties has also been performed to test the performance of MODPSO in this type of problem. The results indicate that the MODPSO shows better performance in finding Pareto solutions and in exploring extreme solutions.

Besides validation of the algorithm, the integrated optimisation approach also needs to be tested and compared to the regular sequential optimisation

approach. This is important to validate the advantages of the integrated optimisation approach for ASP and ALB. It will also identify the classes of problems that are suitable for integrated and sequential optimisation approaches.

The performance of MODPSO has so far only been tested with generated test problems that have never been used in any other works. This chapter therefore aims to validate the performance of the proposed MODPSO algorithm using artificial test problems from the literature and also real-world problems. Other than that, this chapter will also compare the integrated and sequential optimisation approaches numerically. Therefore, this chapter attempts to achieve the following goals:

- Explain the problem selection for validation.
- Validate the proposed MODPSO using artificial problems from the literature.
- Validate the proposed MODPSO using real-world problem.
- Validate the integrated optimisation approach compared to the sequential optimisation approach.

8.1 Problem Selection

For the purpose of validation, a set of problems used in existing publications covering a different range of problem difficulties needs to be selected. In Chapter 5, the assembly problem difficulties have been categorised as low, medium and high difficulties based on the number of tasks, Order Strength, Time Variability ratio and Frequency Ratio. However, since the number of existing problems which consider all of these attributes in their published works is limited, only the number of tasks is considered in order to differentiate the problem difficulties. The usage of the number of tasks to differentiate the

problem difficulties is sufficient because it has great influence on the problem difficulty (Scholl, 1993; Bhattacharjee and Sahu, 1990; Mastor, 1970).

In order to select the problems that represent all the difficulty levels, the test problems from the literature are explored. So far, three problems have been identified for low, medium and high difficulties according to the number of tasks. Additionally, a mixed-model problem has also been selected to be used for validation. These problems are:

i. 20 Task Problem (Chen et al., 2002)

ii. 45 Task Problem (Tseng et al., 2008)

iii. 140 Task Problem of Nissan Pathfinder Engine Assembly (Bautista and Pereira, 2007)

iv. 40 Task Mixed-Model Problem (Tambe, 2006)

The first problem is 20 task problems, adopted from Chen et al. (2002). This problem is selected to represent the low difficulty level. In the meantime, the second problem, with 45 tasks, is selected to represent the medium difficulty level. This problem is cited from Tseng et al. (2008). In the original sources, both problems are used to test the integrated ASP and ALB optimisation using their proposed GA-based algorithms.

Meanwhile, the third problem, selected to represent the high difficulty level, is adopted from Bautista and Pereira (2007). This problem is a real-world problem from Nissan Pathfinder engine assembly, based in Barcelona, Spain. Although the original problem presented in Bautista and Pereira (2007) only considers the ALB optimisation, the problem is selected because of the limited availability of integrated ASP and ALB problems, besides the attraction of real-world problems from industry. Therefore, to adapt the original problem with this work, the ASP data for this problem is randomly generated.

In addition to these three problems, another problem that represents the mixed-model ASP and ALB is also considered. Since there is no existing published work on integrated mixed-model ASP and ALB, a mixed-model ALB problem is

considered to be used for validation. The selected problem is a mixed-model ALB with 40 tasks, originally presented in a thesis (Tambe, 2006). The ASP data for this problem is also randomly generated to adapt it to the integrated mixed-model ASP and ALB optimisation.

So far, the selected problems can be classified into two categories, artificial problems and real-world problems. From the selected problems, problems (i), (ii) and (iv) are artificial problems, while problem (iii) is the real-world problem. To validate the proposed algorithm with the real-world problems, the remaining problems are established from the real assembly products, because of the dearth of literature that presented complete problem information. This process is started by selecting the real assembly products, and followed by identifying the assembly relation between parts. Next, the assembly direction and tool for each task are established by inspecting the actual direction and tool for particular assembly task. Finally, the assembly time is established from the estimation of the author.

The real-world problems which have been established from actual assembly products are listed as follows:

i. Assembly of fixed table vice (12 tasks)
ii. Assembly of toy train (40 tasks)
iii. Assembly of mixed-model table vice (12-19 tasks)

The first real-world problem is assembly of a fixed table vice that represents the low difficulty problem. This product is mainly used in workshops to clamp the work-piece to allow work to be performed on it. The second problem is assembly of a toy train with 40 assembly tasks. This problem represents the medium difficulty problem. Finally, the third problem represents the mixed-model assembly problem. This is the assembly problem of a mixed-model table vice which consists of three models. Each model is designed with different features with the assembly task varying between 12 and 19 tasks.

From the description of the problem selection above, we can summarise the validation problems according to artificial problem and real-world problem categories.

The artificial problems consist of:

1) 20 Task Problem (Chen et al., 2002)
2) 45 Task Problem (Tseng et al., 2008)
3) 40 Task Mixed-Model Problem (Tambe, 2006)

While the real-world problems are:

1) Assembly of fixed table vice (12 tasks)
2) Assembly of toy train (40 tasks)
3) Assembly of Nissan Pathfinder Engine Assembly (140 tasks)
4) Assembly of mixed-model table vice (12-19 tasks)

The results acquired by MODPSO algorithms will be compared to the results presented in the original articles. By using this approach, the performance of MODPSO algorithms can be validated with the best results from the literature. Following that, the performance of the MODPSO algorithm will also be compared with the following comparison algorithms as used in Chapters 6 and 7.

i. Multi-objective Genetic Algorithm (MOGA)
ii. Ant Colony Optimisation (ACO)
iii. Hybrid Genetic Algorithm (HGA)
iv. Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II)
v. Multi-Objective Particle Swarm Optimisation (MOPSO) and
vi. Discrete Particle Swarm Optimisation (DPSO)

In order to analyse and compare the results, the following performance indicators are used (Deb, 2001; Yoosefelahi et al., 2012).

i. Number of non-dominated solution in Pareto optimal, $\tilde{n}$
ii. Error Ratio, *ER*

iii. Generational Distance, *GD*
iv. *Spacing*
v. Maximum Spread, $Spread_{max}$

The details of these performance indicators are available in Section 5.4.2.

8.2 Artificial Problems from Literature

The first section will validate the proposed MODPSO algorithm with the artificial problems that have been used in the selected literature. For this purpose, the evaluation part of the MODPSO algorithm will be altered according to the original problem sources. As mentioned in Section 8.1, three selected artificial problems from the literature are as follows.

1) 20 Task Problem (Chen et al., 2002)
2) 45 Task Problem (Tseng et al., 2008)
3) 40 Task Mixed-Model Problem (Tambe, 2006)

8.2.1 20 Tasks Problem (Chen et al., 2002)

This problem, consisting of 20 tasks, is presented by Chen et al. (2002). In the original paper, a Hybrid Genetic Algorithm is used to optimise this problem. The precedence graph and assembly data is presented in Figure 8.1 below.

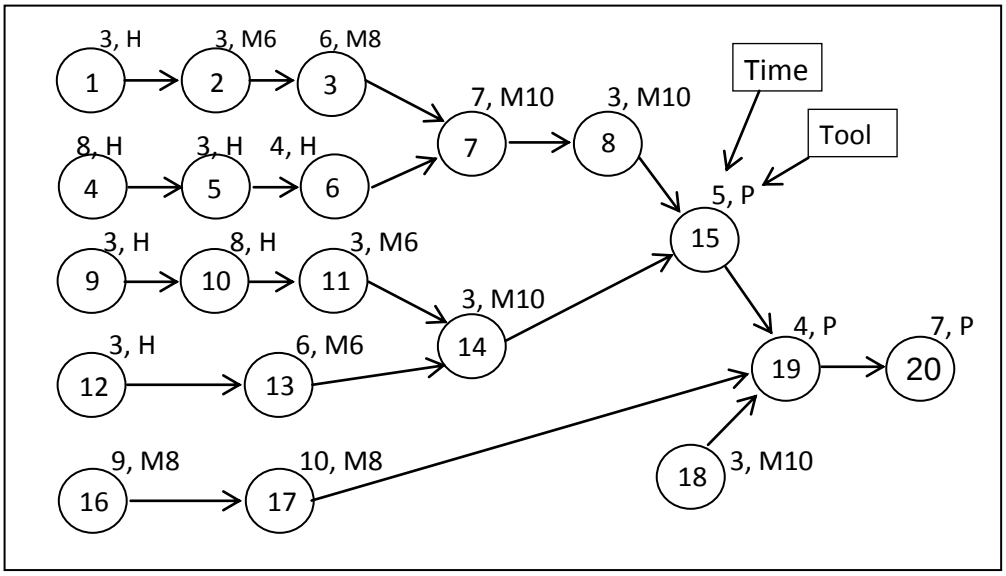


Figure 8.1: Precedence diagram of 20 task problem (Chen et al., 2002)

In this work, five objective functions are used. The objective functions are to minimise:

$f_1 = ct$

ct is cycle time that is defined as the longest processing time in any workstation.

$f_2 = wd$

wd is workload variation. For workstation i=1,2,…,m and i^{th} processing time (pt_i), the wd is calculated as follows:

$$wd = \frac{\sum_{i=1}^{m} (ct - pt_i)}{m}$$

Eq. 8.1

$f_3 = ft$; ft is frequency of tool changes

$f_4 = tn$; tn is the total number of tools that are required in all workstations.

$f_5 = tp$; tp is the total assembly complexity that are calculated based on penalty index matrix.

The original article used population size between 30 and 60. The original article also presented the result that was acquired from only one run. Therefore, the MODPSO and other comparison algorithms use 30 particles and run the experiment only one round.

Optimisation Results

The performance indicators for all algorithms are presented in Table 8.1. The numbers in brackets are weighting values that are assigned to each algorithm based on its performance for the respective indicator. For every indicator, the best result is assigned a weight value of 7, followed by 6 for second best, 5 for third best and so on. The summation of weight values is then calculated to determine the overall algorithm performance. Based on the summation of assigned weight, the algorithm performance rank is then determined.

Table 8.1: Summary of performance indicators of 20 tasks problem

Algorithm	$\tilde{\eta}$[1]	ER[2]	GD[2]	$Spacing$[2]	$Spread_{max}$[1]	Weight sum	Rank
MOGA	0 (4)	1.0000 (3)	2.6013 (4)	1.6797 (7)	18.3737 (2)	20	4
ACO	0 (4)	1.0000 (3)	3.7627 (1)	3.3192 (1)	24.9957 (5)	14	7
HGA	6 (6)	0.7000 (6)	1.6324 (6)	2.2544 (3)	20.0445 (3)	24	2
NSGA-II	1 (5)	0.8889 (5)	1.8447 (5)	1.6915 (6)	11.6014 (1)	22	3
MOPSO	1 (5)	0.9545 (4)	3.4781 (2)	3.0763 (2)	32.6381 (7)	20	4
DPSO	0 (4)	1.0000 (3)	3.4517 (3)	2.0561 (4)	27.8646 (6)	20	4
MODPSO	26 (7)	0.2353 (7)	0.3595 (7)	1.7688 (5)	23.7959 (4)	30	1

[1] Larger the better indicator [2] Smaller the better indicator

*Number in brackets are weighting values from the best (weight=7) to worst (weight=1)

Based on Table 8.1, the MODPSO algorithm shows better performance compared to other algorithms in $\tilde{\eta}$, ER and GD indicators. Meanwhile, for this same set of performance indicators, the HGA algorithm used in the original article is in the second place. On the other hand, for *Spacing* indicator, MOGA algorithm shows better performance, followed by NSGA-II and MODPSO

algorithms. Finally, the MOPSO algorithm performed better for the $Spread_{max}$ indicator, while the MODPSO algorithm dropped to fourth place.

Based on the summation of weight, the proposed MODPSO algorithm demonstrates the best overall performance, followed by HGA as used in the original article. This result indicates that the proposed algorithm is able to perform better compared to comparable algorithms, including the HGA, especially in finding the Pareto optimal solutions.

In order to compare the performance of the proposed MODPSO to the results published in Chen et al. (2002), the non-dominated solution presented in the original article and from MODPSO is presented in Table 8.2 and Table 8.3.

Table 8.2: Non-dominated solutions from original article (Chen et al., 2002)

No.	f_1	f_2	f_3	f_4	f_5	Sequence
a.	20	1.7	3	9	4	9-10-18-14-16-17-15-19-20-12-11-2-13-1-4-5-6-3-7-8
b.	20	2.2	2	8	7	1-4-5-2-6-12-9-10-7-8-18-14-3-13-11-16-17-15-19-20
c.	21	2.2	4	10	0	9-10-18-14-16-17-11-15-20-2-13-19-1-4-5-6-12-3-7-8
d.	21	3.2	2	8	2	9-10-18-14-16-17-15-19-20-2-13-11-1-4-5-6-12-3-7-8

The result shows that solution (b) in Table 8.2 is dominated by solutions 15, 25 and 26 from the MODPSO algorithm in Table 8.3. The remaining three solutions are Pareto optimal, as shown in Table 8.2. The MODPSO algorithm failed to find these solutions since the MODPSO is purposely designed to generate only feasible assembly sequences. In this case, the remaining three solutions (solutions a, c and d) have violated the precedence constraint, because task 14 can only be performed once tasks 11 and 13 are completed (refer to Figure 8.1). In all three solutions, task 14 is assigned before both tasks 11 and 13.

Table 8.3: Non-dominated solutions from MODPSO algorithm

No.	f_1	f_2	f_3	f_4	f_5	Sequence
1	14	1.4	7	14	15	1-2-3-4-5-12-13-6-9-7-18-8-10-11-14-16-15-17-19-20
2	14	1.4	9	16	9	1-2-3-4-5-12-13-18-6-7-8-9-10-11-14-16-15-17-19-20
3	15	2.4	6	13	11	4-12-16-9-10-1-2-13-11-18-14-3-5-6-7-8-15-17-19-20
4	15	2.4	8	14	3	1-2-9-18-10-12-16-13-3-11-14-4-5-6-7-8-15-17-19-20
5	16	1.6	1	8	13	1-16-4-5-9-10-6-12-13-11-2-17-3-7-8-18-14-15-19-20
6	16	1.6	2	9	11	1-9-10-4-5-12-13-11-2-6-3-16-7-18-17-8-14-15-19-20
7	16	1.6	3	10	9	1-12-9-2-10-13-4-5-6-3-16-17-11-7-8-18-14-15-19-20
8	16	1.6	4	10	7	1-2-12-9-4-5-6-3-16-10-11-17-13-7-8-18-14-15-19-20
9	16	1.6	6	11	6	1-2-9-18-4-5-6-3-16-10-12-17-13-7-8-11-14-15-19-20
10	16	1.6	7	12	5	9-18-1-2-12-3-16-10-13-4-5-6-7-8-17-11-14-15-19-20
11	16	1.6	8	14	4	1-2-9-18-4-5-12-13-16-17-6-10-11-14-3-7-8-15-19-20
12	17	2.6	1	8	9	1-9-10-12-11-2-13-4-5-6-3-16-7-14-17-8-18-15-19-20
13	17	2.6	5	11	3	1-2-9-12-4-5-6-3-13-10-16-11-17-7-8-18-14-15-19-20
14	17	2.6	8	13	2	18-9-10-11-1-2-4-5-6-12-13-14-3-16-7-8-15-17-19-20
15	18	1.2	2	8	7	1-12-9-10-2-13-16-3-17-4-5-6-11-7-8-18-14-15-19-20
16	18	1.2	5	10	5	1-2-9-10-4-5-6-11-12-13-16-3-17-7-8-18-14-15-19-20
17	18	1.2	6	11	4	1-9-16-10-17-2-3-12-13-4-5-6-11-7-8-18-14-15-19-20
18	18	1.2	6	12	2	1-9-10-11-12-13-16-2-4-5-6-18-14-17-3-7-8-15-19-20
19	18	1.2	8	12	1	1-2-9-10-3-18-16-4-12-13-5-6-7-8-11-14-17-15-19-20
20	18	1.2	9	14	0	9-18-16-4-12-13-10-17-1-2-3-11-14-5-6-7-8-15-19-20
21	18	3.6	4	9	6	1-9-12-10-11-2-3-13-4-5-6-7-8-18-14-16-15-17-19-20
22	18	3.6	6	10	4	1-2-12-13-3-16-4-5-6-9-18-7-8-10-17-11-14-15-19-20
23	18	3.6	7	11	2	1-2-18-12-13-9-10-11-4-5-6-3-16-7-8-14-15-17-19-20
24	18	3.6	8	13	0	1-2-9-18-4-12-13-3-16-10-17-5-6-7-8-11-14-15-19-20
25	19	2.2	1	7	6	1-9-10-12-2-13-11-3-4-5-6-16-17-7-8-18-14-15-19-20
26	19	2.2	2	8	4	1-9-10-12-2-13-11-4-5-6-16-17-3-7-8-18-14-15-19-20
27	19	2.2	4	9	2	1-9-10-12-11-2-18-13-4-5-6-14-16-17-3-7-8-15-19-20
28	19	2.2	5	10	0	1-9-10-11-4-5-6-12-18-13-14-16-17-2-3-7-8-15-19-20
29	19	4.6	5	9	0	1-9-10-12-2-3-11-16-17-18-13-14-4-5-6-7-8-15-19-20
30	21	0.7	7	12	17	16-1-9-12-17-10-2-3-13-4-5-6-11-14-7-8-15-18-19-20
31	22	0.7	9	12	11	9-16-1-17-4-2-5-12-13-10-11-6-3-7-8-18-14-15-19-20
32	22	0.7	9	14	8	9-16-12-17-1-2-3-18-13-4-10-11-5-6-7-8-14-15-19-20
33	22	0.7	9	15	5	9-18-16-17-1-2-3-12-13-4-10-11-5-6-7-8-14-15-19-20
34	23	0.7	11	13	10	9-16-12-13-10-11-1-2-17-4-3-5-18-6-7-8-14-15-19-20

The result from the MODPSO algorithm offers alternative feasible solutions for solutions (a), (c) and (d). For solution (a), the alternative solution from Table 8.3 is solution 15, compromising the f_5 objective. Another possible alternative for solution (a) is solution 26 but with higher f_2. Meanwhile for solution (c), the MODPSO results propose an alternative of solution 28 with trade-off of f_3, but in return offering better f_1. Finally, the alternative solutions for solution (d) will be solutions 5, 12 and 26 by compromising f_5 objective.

The alternative solutions for solutions (a), (c) and (d) not only offer the nearest non-dominated solution found using the MODPSO algorithm, but also feasible assembly sequences that fulfilled the precedence constraint. In a real assembly process, it is very important to follow this constraint because the product cannot be assembled correctly when the precedence constraint is violated.

8.2.2 45 Tasks Problem (Tseng et al., 2008)

This integrated ASP and ALB problem is presented in Tseng et al. (2008). This problem originated from the famous Kilbridge and Wester (1961) problem for ALB that was widely used to test algorithms. On top of the original problem, Tseng has randomly generated the assembly tool and direction data to test their Hybrid Evolutionary Multiple-Objective Algorithm (HEMOA). The precedence graph and assembly data is presented in Figure 8.2 and Table 8.4.

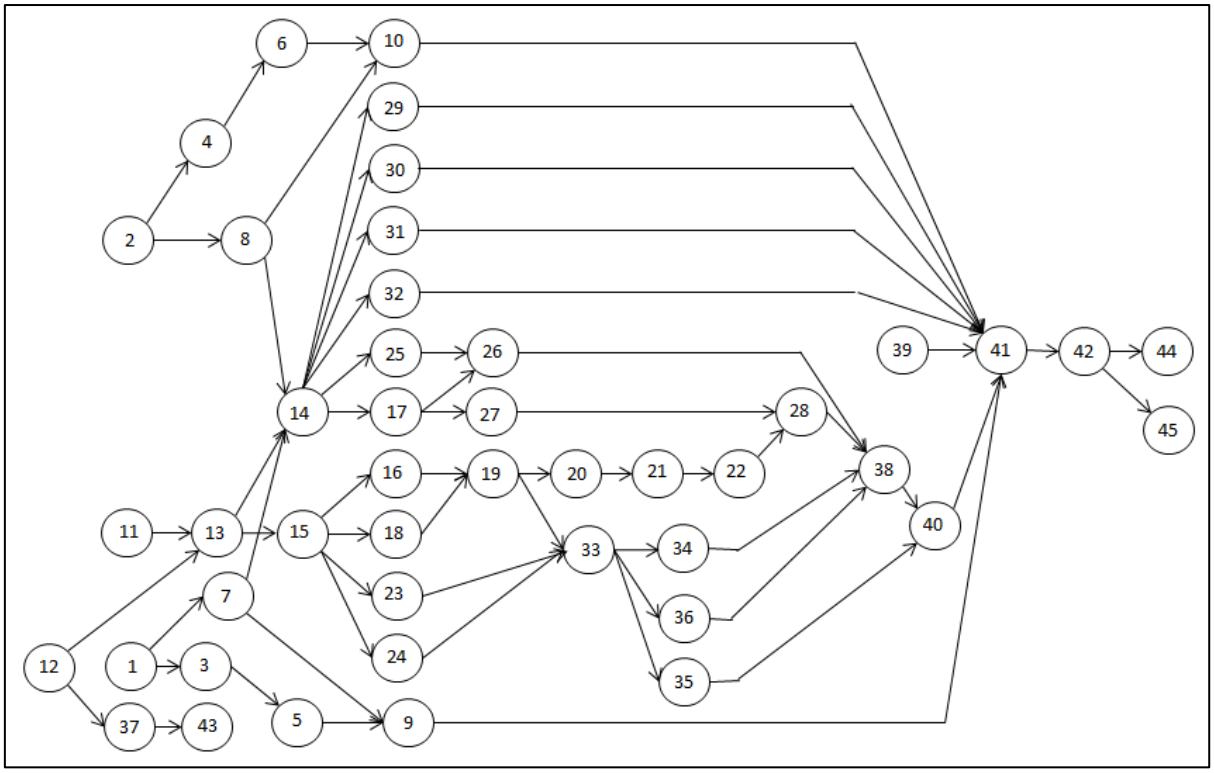


Figure 8.2: Precedence diagram of 45 task problem (Tseng et al., 2008)

Table 8.4: Assembly data of 45 task problem

Task	*D*	*T*	*M*	Task	*D*	*T*	*M*
1	*-y*	T3	9	24	*x*	T4	27
2	*-z*	T2	9	25	*y*	T1	29
3	*y*	T4	10	26	*x*	T3	26
4	*-z*	T1	10	27	*-y*	T1	6
5	*-z*	T4	17	28	*z*	T4	5
6	*-y*	T1	17	29	*-z*	T3	24
7	*z*	T4	13	30	*-x*	T3	4
8	*-y*	T4	13	31	*z*	T4	5
9	*y*	T2	20	32	*x*	T1	7
10	*-z*	T4	20	33	*-z*	T3	4
11	*y*	T3	10	34	*z*	T2	15
12	*-x*	T4	11	35	*z*	T2	3
13	*x*	T3	6	36	*x*	T4	7
14	*-y*	T4	22	37	*-x*	T4	9
15	*x*	T2	11	38	*z*	T3	4
16	*-z*	T1	19	39	*x*	T3	7
17	*y*	T1	12	40	*x*	T1	5
18	*-z*	T2	3	41	*z*	T2	4
19	*-z*	T1	7	42	*-z*	T3	21
20	*z*	T1	4	43	*y*	T1	12
21	*-z*	T3	55	44	*-y*	T4	6
22	*-z*	T1	14	45	*-x*	T3	5
23	*-y*	T3	9				

In Tseng's work, three objective functions were used for optimisation purposes. The first objective (f_1) is to minimise assembly direction change and the second objective (f_2) is to minimise the assembly tool change. For m is the number of workstation and N_i is the number of tasks in i^{th} workstation, the f_1 and f_2 are calculated as follows:

$$\min f_1 = \sum_{i=1}^{m} \sum_{n=2}^{N_i} DC_n$$

Eq. 8.2

$$DC_n = \begin{cases} 0, & \text{if the direction } n^{th} \text{ task is similar to the previous task} \\ 1, & \text{if the rotation angle is } 90° \\ 2, & \text{if the rotation angle is } 180° \end{cases}$$

$$\min f_2 = \sum_{i=1}^{m} \sum_{n=2}^{N_i} TC_n$$

Eq. 8.3

$$TC_n = \begin{cases} 0, & \text{if the assembly tool of } n^{th} \text{ task is similar to the previous task} \\ 1, & \text{if the assembly tool of } n^{th} \text{ task is not the same as the previous task} \end{cases}$$

Meanwhile, the third objective (f_3) is to minimise the workload difference among workstations.

$$\min f_3 = \text{Round} \left[\sqrt{\sum_{i=1}^{m} \left(\frac{100(T_i - T_c)}{T_c} \right)^2} \right]$$

Eq. 8.4

In this case, T_i is total assembly time for workstation i and T_c is expected cycle time that was calculated by dividing the total assembly time for all tasks by the total number of workstations.

Optimisation Results

For experiment purposes, the number of particles in MODPSO is set as 50 and the maximum iteration is 500 as used in Tseng's original article. A similar setting is also applied in all comparison algorithms. The summary of performance indicators including the HEMOA, presented in the original article, is presented in Table 8.5. For this problem, the assigned weight (number in brackets) ranges from 1 to 8, since there are 8 sets of results (including the HEMOA as presented in the original article) to be considered.

From Table 8.5, the proposed MODPSO algorithm performed better in terms of finding the Pareto optimal solutions represented by indicator *ñ*. For indicator *ER* and *GD*, the NSGA-II performed better, while in the *Spacing* indicator, the HGA performs better compared to other algorithms. For *ER*, *GD* and *Spacing*

indicators, the proposed MODPSO consistently performed in second position behind the leaders.

Table 8.5: Comparison of performance indicators for 45 tasks problem

Algorithm	$\tilde{\eta}^1$	ER^2	GD^2	$Spacing^2$	$Spread_{max}{}^1$	Weight sum	Rank
MOGA	9 (6)	0.6087 (6)	3.3836 (6)	5.9338 (2)	98.5748 (4)	24	4
ACO	0 (3)	1.0000 (2)	7.9469 (2)	5.1322 (4)	107.3965 (6)	17	7
HGA	2 (4)	0.9355 (3)	5.5043 (4)	2.8701 (8)	92.3309 (3)	22	5
NSGA-II	13 (7)	0.1333 (8)	0.5399 (8)	4.5588 (6)	65.4370 (1)	30	2
MOPSO	2 (4)	0.9310 (4)	6.8064 (3)	6.2054 (1)	126.0555 (7)	19	6
DPSO	0 (3)	1.0000 (2)	4.2639 (5)	5.4114 (3)	85.3288 (2)	15	8
MODPSO	25 (8)	0.3056 (7)	1.5315 (7)	3.3979 (7)	107.1588 (5)	34	1
HEMOA	8 (5)	0.7895 (5)	1.5315 (7)	4.9052 (5)	209.3848 (8)	30	2

[1] Larger the better indicator [2] Smaller the better indicator

*Number in brackets are weighting values from the best (weight=8) to worst (weight=1)

Finally for the $Spread_{max}$ indicator, the algorithm that was used in the original article performed better compared to other algorithms. This result indicates that the MODPSO algorithm performs better in finding Pareto solutions, while the HEMOA demonstrates better ability in exploring the search space. On the other hand, the NSGA-II presents better accuracy towards Pareto solutions compared to other algorithms. The overall performance of the algorithms to optimise this problem is summarised by summation of assigned weight in the second last column. Based on the weight summation, the proposed MODPSO is in the first rank, followed by NSGA-II and HEMOA in the second rank.

In comparing the performance of MODPSO and HEMOA, the MODPSO came out with a larger number of found Pareto solutions although the HEMOA presents more non-dominated solutions in the original article. The non-dominated solutions, as presented in the original paper, are shown in Table 8.6, while the non-dominated solutions acquired from MODPSO are in Table 8.7.

From 40 non-dominated solutions in Table 8.6, only 8 (Table 8.5) are Pareto optimal, while the remaining solutions are dominated by at least one solution

from other algorithms. Meanwhile, from 36 non-dominated solutions from MODPSO, 25 of them are Pareto optimal.

Table 8.6: Non-dominated solutions from HEMOA (Tseng et al., 2008)

No	f_1	f_2	f_3	No.	f_1	f_2	f_3	No.	f_1	f_2	f_3	No.	f_1	f_2	f_3
1	19	20	223	**11**	25	16	151	**21**	27	19	25	**31**	30	15	59
2	20	18	218	**12**	25	18	73	**22**	27	20	15	**32**	30	16	35
3	21	19	175	**13**	25	19	71	**23**	28	9	119	**33**	30	17	24
4	22	21	146	**14**	26	10	108	**24**	28	12	73	**34**	31	11	81
5	23	20	140	**15**	26	16	64	**25**	28	17	58	**35**	31	15	55
6	23	24	133	**16**	26	23	15	**26**	28	18	15	**36**	32	16	34
7	23	25	90	**17**	27	12	92	**27**	29	8	132	**37**	33	13	72
8	24	18	153	**18**	27	13	66	**28**	29	9	102	**38**	33	14	65
9	24	20	126	**19**	27	15	73	**29**	29	16	49	**39**	34	12	70
10	25	15	152	**20**	27	18	56	**30**	30	11	96	**40**	36	10	90

Table 8.7: Non-dominated solutions from MODPSO algorithm

No.	f_1	f_2	f_3	No.	f_1	f_2	f_3	No.	f_1	f_2	f_3	No.	f_1	f_2	f_3
1	18	17	77	**10**	22	21	26	**19**	24	17	19	**28**	25	20	15
2	18	18	59	**11**	22	26	25	**20**	24	18	18	**29**	26	12	118
3	19	15	55	**12**	22	29	22	**21**	24	25	17	**30**	27	12	106
4	19	16	46	**13**	23	14	88	**22**	24	26	13	**31**	27	13	48
5	21	22	31	**14**	23	18	23	**23**	25	13	93	**32**	27	14	26
6	21	27	24	**15**	23	20	19	**24**	25	14	58	**33**	28	12	49
7	22	14	114	**16**	23	28	17	**25**	25	15	26	**34**	28	18	15
8	22	17	45	**17**	24	14	65	**26**	25	16	20	**35**	28	25	14
9	22	20	34	**18**	24	16	26	**27**	25	17	16	**36**	31	23	14

In addition, the original article also presents the best line balancing result using the HEMOA with f_3 equal to 15, as shown by solution 26 in Table 8.6. The MODPSO algorithm is able to find assembly task assignment with better f_3, which is equal to 13, as presented by solution 22 in Table 8.7. The comparison of best task assignment from Tseng's article and MODPSO algorithm for line balancing objective (f_3) are presented in Table 8.8.

This result shows that the MODPSO was able to find better balancing for Kilbridge and Wester's problem compared with HEMOA. It also shows that although the MODPSO was designed for integrated ASP and ALB, this algorithm was still capable of finding a better solution for the famous Kilbridge and Wester problem which was specifically designed for ALB problem.

Table 8.8: Comparison of the best line balancing results between HEMOA and MODPSO

From Tseng et al. (2008)			MODPSO		
i	Task Assignment	T_i	i	Task Assignment	T_i
1	[1 3 5 7]	49	1	[12 1 2 11 3]	49
2	[2 9 11 4]	49	2	[7 39 37 5 13]	52
3	[6 8 12 37]	50	3	[15 24 4]	48
4	[13 15 18 14 32]	49	4	[6 8 14]	52
5	[24 39 30 23 31]	52	5	[30 9 31 17 23]	50
6	[25 29]	53	6	[25 16 18]	51
7	[16 19 20 10]	50	7	[43 10 19 20 27]	49
8	[21]	55	8	[21]	55
9	[17 22 27 33 28 36]	48	9	[33 22 36 26]	51
10	[26 34 35 38]	48	10	[28 34 29 38 35]	51
11	[43 40 41 42 45 44]	53	11	[32 40 41 42 45 44]	48
f_3 = 15			f_3 = 13		

*For T_c=50.5; f_3 is calculated using Eq. 8.4

8.2.3 40 Tasks Mixed-Model Assembly Problem (Tambe, 2006)

This problem is adopted from Tambe's thesis in 2006; which consists of 40 tasks for Mixed-Model ALB. Since there are no existing published works on integrated mixed-model ASP and ALB, the ASP data (i.e. assembly direction and tool) is generated using a test problem generator, while the ALB data (i.e. assembly time) is adopted from the original problem (Tambe, 2006). The joint precedence diagram for this problem is presented in Figure 8.3, while the assembly data for Model A, B and C is in Table 8.9.

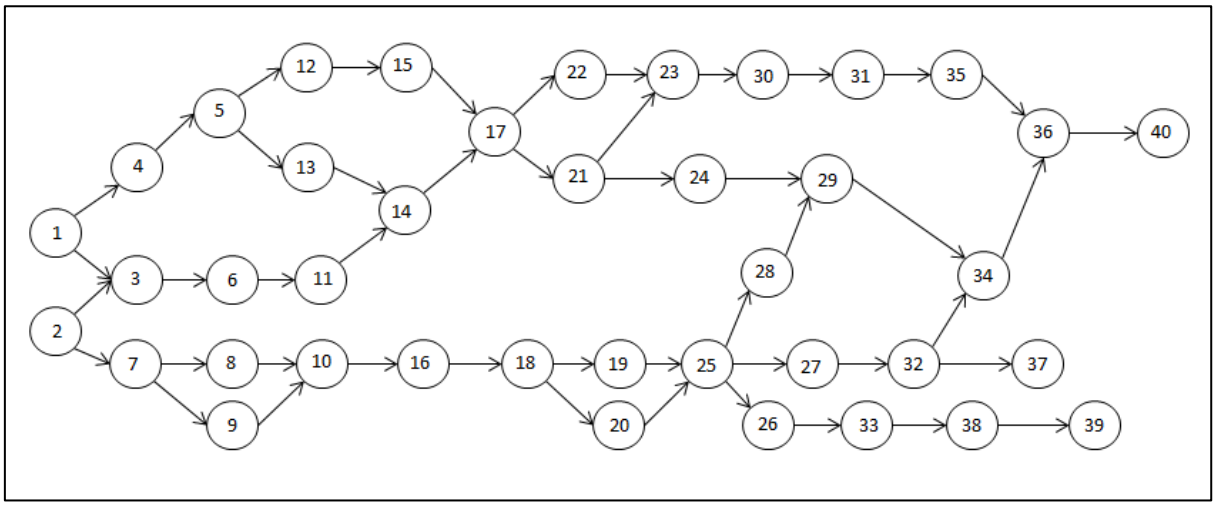


Figure 8.3: The joint precedence diagram of 40 tasks mixed-model problem (Tambe, 2006)

In the original work, which only considers the ALB, a novel heuristic approach is used to solve and optimise the mixed-model ALB. The optimisation objective is to minimise the number of workstations for given cycle time and to minimise the variation time. The variation time (also known as idle time) is the difference between cycle time and time allotted per workstation. The cycle time for each model is 30 time units with flexibility factor 1.05. It means that the maximum cycle time can be increased up to 31.5 time units (Tambe, 2006). By using the given cycle time, the total assembly time for all models in each workstation is 90 time units.

Table 8.9: Assembly data of 40 tasks mixed-model problem

Task	Model A			Model B			Model C			Task	Model A			Model B			Model C		
	D	*T*	*M*	*D*	*T*	*M*	*D*	*T*	*M*		*D*	*T*	*M*	*D*	*T*	*M*	*D*	*T*	*M*
1	4	1	10	4	1	12	1	4	13	21	2	1	8	3	3	6	4	2	6
2	4	4	8	5	1	9	6	4	10	22	4	3	11	3	4	10	3	5	11
3	3	3	6	5	3	6	3	1	6	23	3	2	6	5	1	4	1	5	4
4	4	4	12	3	4	15	3	4	17	24	2	1	16	5	1	12	5	4	13
5	6	3	10	1	4	8	1	2	9	25	1	5	12	6	3	11	4	1	12
6	1	2	15	2	4	12	4	3	13	26	3	2	6	3	3	4	1	1	4
7	2	2	10	4	5	8	1	5	9	27	5	2	8	3	5	6	1	1	6
8	2	2	6	1	4	5	1	2	5	28	5	1	11	5	3	9	6	2	10
9	3	4	8	4	1	10	1	1	11	29	1	1	12	3	3	9	4	3	10
10	6	5	14	4	5	11	3	3	12	30	6	4	7	6	5	5	3	5	5
11	4	1	11	3	4	8	2	3	9	31	4	4	5	1	3	6	1	4	6
12	3	5	13	1	5	12	2	4	13	32	6	1	11	2	4	8	4	4	9
13	6	4	18	2	4	15	4	1	17	33	4	5	13	3	1	10	6	1	11
14	2	4	6	6	4	4	6	2	4	34	1	3	11	5	3	12	4	1	13
15	4	3	15	6	5	12	3	1	13	35	5	2	9	5	1	7	1	5	8
16	2	5	14	2	3	16	5	5	18	36	4	2	6	2	5	9	1	2	10
17	1	1	13	6	1	10	1	1	11	37	1	4	17	1	5	13	2	5	14
18	5	1	9	2	5	7	2	4	8	38	4	4	12	6	5	9	1	5	10
19	4	4	7	4	5	9	6	4	10	39	4	2	14	4	3	14	4	5	16
20	5	5	12	6	3	9	4	4	10	40	1	3	6	2	4	4	6	4	4

Optimisation Results

The experiment of this problem is performed using the proposed MODPSO algorithm. On top of the original ALB objectives, this experiment also considers the ASP objectives. The optimisation objectives are:

i. Minimise total direction change
ii. Minimise total tool change
iii. Minimise allotted time
iv. Minimise number of workstations
v. Minimise total variations

For each algorithm, the population size and maximum iteration are 20 and 500, while 10 repetitions with different random seeds are used. The performance indicators for this problem are presented in Table 8.10.

Table 8.10: Comparison of performance indicators for mixed-model problem

Algorithm	$\tilde{\eta}$[1]	*ER*[2]	*GD*[2]	*Spacing*[2]	$Spread_{max}$[1]	Weight sum	Rank
MOGA	6 (5)	0.8462 (5)	1.1967 (5)	0.6246 (5)	8.3453 (3)	23	3
ACO	0 (2)	1.0000 (2)	2.1298 (1)	0.6882 (4)	7.3911 (1)	10	7
HGA	1 (3)	0.9815 (3)	1.4290 (4)	0.6198 (6)	9.2722 (4)	20	5
NSGA-II	19 (6)	0.0500 (7)	0.0687 (7)	0.7978 (2)	8.0958 (2)	24	2
MOPSO	4 (4)	0.9273 (4)	1.4928 (3)	0.5735 (7)	10.866 (5)	23	3
DPSO	0 (2)	1.0000 (2)	1.6578 (2)	0.8133 (1)	11.0538 (6)	13	6
MODPSO	43 (7)	0.6228 (6)	0.7856 (6)	0.7283 (3)	13.5221 (7)	29	1

[1] Larger the better indicator [2] Smaller the better indicator

*Number in brackets are weighting values from the best (weight=7) to worst (weight=1)

Based on the performance indicators in Table 8.10, the MODPSO algorithm performed better in $\tilde{\eta}$ and $Spread_{max}$ indicators compared to all other algorithms. The MODPSO algorithm is able to find 43 out of 73 Pareto solutions. Meanwhile the NSGA-II performed well, by leading the *ER* and *GD* indicators, while MODPSO was in second place. On the other hand, the MOPSO algorithm shows better performance for *Spacing* indicator. For this problem, the proposed MODPSO algorithm once again shows better ability in finding Pareto solutions, while the NSGA-II presents better accuracy towards Pareto solutions compared to other algorithms. This result is also aligned with earlier findings in Chapter 7 which concluded that the MODPSO shows significant improvement over comparable algorithms for $\tilde{\eta}$ and $Spread_{max}$ indicators.

For this problem, the proposed MODPSO also shows better overall performance compared to other algorithms according to summation of weight. This result put the MODPSO in the first rank, followed by NSGA-II in the second rank, while the MOGA and MOPSO share the third rank.

The original work presents the best mixed-model ALB result with 15 workstations and 166 time units of variation, as shown in Table 8.11 (Tambe, 2006). In workstation 10, the total assembly time for all models (Allotted time) is larger than the given cycle time (i.e. 90 time unit), because of the flexibility factor, as explained earlier. Although the largest allotted time is 92, the cycle time for this solution is 90 time units because the predetermined maximum

cycle time is 90 and the allotted time in workstation 10 is acquired from the flexibility factor (Tambe, 2006).

Table 8.11: Optimum mixed-mode line balancing result from Tambe (2006)

Workstation	Allotted time	Variation
1	83	7
2	88	2
3	72	18
4	71	19
5	75	15
6	85	5
7	78	12
8	82	8
9	86	4
10	92	-2
11	89	1
12	89	1
13	67	23
14	83	7
15	44	46
	Total variation	166

Compared to the results presented in Tambe's thesis, the proposed MODPSO algorithm has successfully assigned assembly tasks into 15 workstations, as in Tambe's work, but with smaller total variation as shown in Table 8.12. The maximum allotted time for this solution is 85 time units, which is better than the 90 time units in Tambe's work. The total variation of this solution is only 91 time units, which shows better balancing for MODPSO's solution. Besides that, in all models of the MODPSO's solution, the predetermined maximum cycle time is fully complied. This is very important to ensure the assembly line is able to fulfil the product demand.

Table 8.12: Optimum mixed-mode line balancing result of MODPSO

Workstation	Tasks	T_A	T_B	T_C	Allotted time	Variation
1	2,7,9	26	27	30	83	2
2	1,4	22	27	30	79	6
3	5,3,12	29	26	28	83	2
4	15,6	30	24	26	80	5
5	11,13	29	23	26	78	7
6	14,17,22	30	24	26	80	5
7	21,24,23	30	22	23	75	10
8	30,31,35,8	27	23	24	74	11
9	10,16	28	27	30	85	0
10	18,19,20	28	25	28	81	4
11	25,28,26	29	24	26	79	6
12	29,33	25	19	21	65	20
13	38,39	26	23	26	75	10
14	27,32,34	30	26	28	84	1
15	37,36,40	29	26	28	83	2
Total variation						91

8.2.4 Summary of Validation using Artificial Test Problems from the Literature

This section presents validation of the proposed MODPSO algorithm with three selected artificial problems from the literature. In the first problem with 20 tasks, the MODPSO algorithm was able to find 26 Pareto solutions compared to the presented solution in Chen's original article which only has 4 solutions (Chen et al., 2002). From these 4 solutions, one of them is dominated by the solution from MODPSO. In contrast with MODPSO which presented all feasible solutions, the remaining 3 Pareto solutions as in the original article are unfeasible because the precedence constraint was violated.

In the second problem with 45 tasks, 32 out of 40 solutions which were presented in Tseng's original article are dominated by another solution from another algorithm (Tseng et al., 2008). From experimental results, only 8 solutions from the original article are Pareto optimal solutions compared to 25 solutions from the MODPSO algorithm. The MODPSO algorithm was also able

to find better minimum fitness in direction change (f_1) for ASP and workload difference (f_3) for ALB compared to results in the article.

Finally, the third problem presents the Mixed-Model ALB with 40 tasks from Tambe's thesis (Tambe, 2006). The ASP data is generated on top of the original ALB problem. The multi-objective optimisation concluded that the MODPSO algorithm performed better than other algorithms in finding Pareto optimal solutions and exploring better extreme solutions. Compared to Tambe's result, the MODPSO is able to assign the tasks with better balancing in a similar number of workstations.

The experimental results from all three problems demonstrate that the proposed MODPSO algorithms were able to find better solutions compared to presented results from the selected published works. This is clearly shown in the problems with 20 and 45 tasks. The MODPSO algorithm also presents better line balancing with similar number of workstations in the mixed-model problem. Besides that, the MODPSO algorithm also shows better overall performance compared with comparison algorithms for all three selected problems. Therefore, through these three test problems, the proposed MODPSO algorithm has been validated to produce better results compared with the results published in the original articles (Chen et al., 2002; Tseng et al., 2008; Bautista and Pereira, 2007; Tambe, 2006).

8.3 Real-World Problems

This section will validate the ability of the proposed MODPSO algorithm to optimise real-world problems. It will explain how the product representation is established, followed by an evaluation example and finally the optimisation of real life problems. Four different real life problems have been identified for use in this section. These problems are:

1) Assembly of fixed table vice
2) Assembly of toy train
3) Assembly of Nissan Pathfinder engine (Bautista and Pereira, 2007)
4) Assembly of mixed-model table vice

Five objective functions are used to optimise these problems. For single-model problems (Problems 1 to 3), the following objectives are used, as in Chapter 6.

f_1 = Minimise number of assembly direction change (n_{dc})

f_2 = Minimise number of assembly tool change (n_{tc})

f_3 = Minimise cycle time(ct)

f_4 = Minimise number of workstations (nws)

f_5 = Minimise workload variation (v)

The details of these objectives are presented in Section 6.3.1. Meanwhile, for the mixed-model problem (Problem 4), the objectives as presented in Section 7.2.1 are applied.

8.3.1 Assembly of Fixed Table Vice

The first example is a fixed table vice consisting of 12 parts. The exploded drawing of this product is shown in Figure 8.4. From this drawing, the assembly liaison is identified and recorded in a liaison matrix. As an example from Table 8.13, part P1 has a relation with part P8 and part P9. Therefore, the liaison matrix, L (P1, P8) = 1 and L (P1, P9) = 2. Next, the relation for part P2 is

identified by inspecting this part with other parts. After all the liaisons are identified, 12 liaisons are recorded in the liaison matrix.

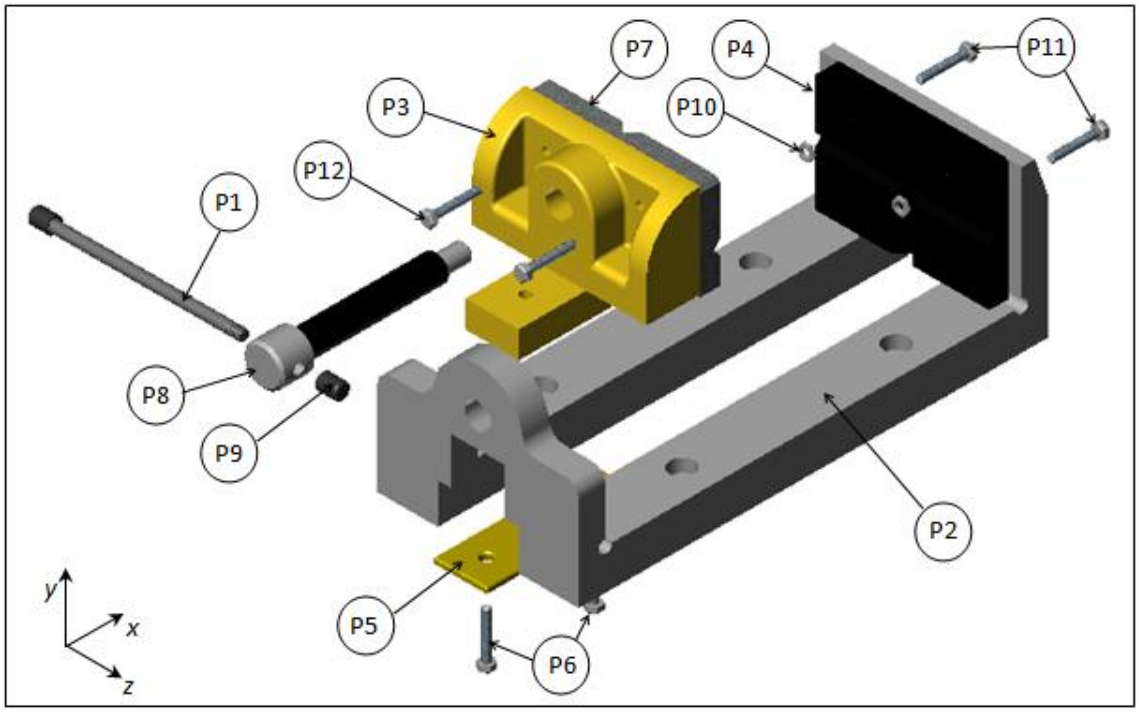


Figure 8.4: Exploded model of table vice

Table 8.13: Liaison matrix for table vice assembly

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
P1	-							1	2			
P2		-	3	4				5			6	
P3			-		7	8	9	10				
P4				-								
P5					-							
P6						-						
P7							-					11
P8								-				
P9									-			
P10										-	12	
P11											-	
P12												-

To establish the precedence constraint set, De Fazio's question and answer procedure is applied. This procedure is adopted from De Fazio and Whitney (1987) which consists of two questions for each task.

For task *i*;

Question 1: What tasks must be done prior to doing the task *i*?

Question 2: What tasks must be left to be done after doing the task *i*?

For this example, De Fazio's question and answer summary is presented in Table 8.14. For example, for task 1, there is no precedence task needs to be done prior to task 1. However, to perform 1, assembly task 2 must be left to be performed after task 1. Otherwise, the assembly task 1 cannot be performed. This constraint is translated into precedence constraint C (1,2) that shows task 1 must be done prior to 2.

Table 8.14: Summary of De Fazio's Q&A for table vice assembly

Task	Answer for		Task	Answer for	
	Q1	Q2		Q1	Q2
1	-	2	7	3	8
2	1	-	8	3, 7	-
3	11	10	9	-	-
4	-	12	10	5	-
5	-	10	11	9	3
6	4	-	12	4, 6	-

The question and answer summary from De Fazio's question is then transformed into the precedence constraint. For this problem, the precedence is given as C [(1,2), (5,8), (3,8), (7,8), (11,3), (3,7), (3,10), (4,6), (4,12), (6,12), (9,11)]. Then the initial precedence graph is mapped as shown in Figure 8.5. The precedence constraint is represented by a directed arc.

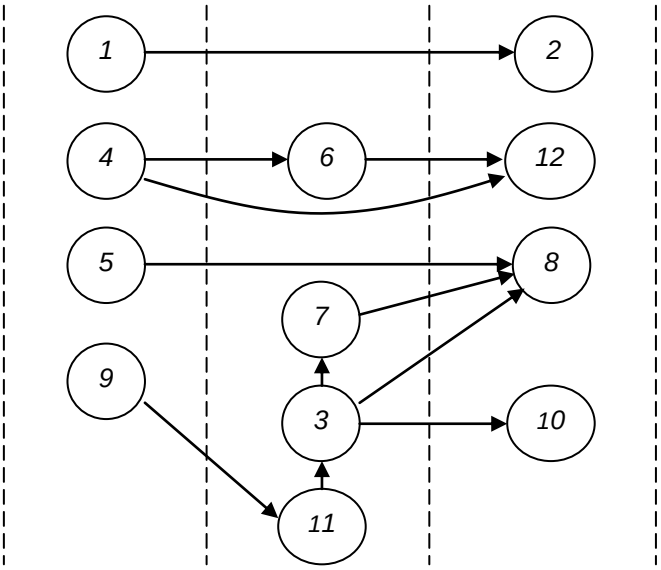


Figure 8.5: Precedence graph mapping for table vice assembly

In this example, there are two generic routes. The first route is from 4 to 12, whereas it also can be reached from 4 to 6 and to 12. Then the second repetitive route is from 3 to 8. The alternative way for this route is to start from 3 to 7 and to 8. For the repetitive route, the shortest route is eliminated from the precedence graph. The assembly data is established as a basis for sequence evaluation. In summary, this assembly process involved all assembly directions (*+x,-x,+y,-y,+z,-z*) and three assembly tools, as presented in Table 8.15.

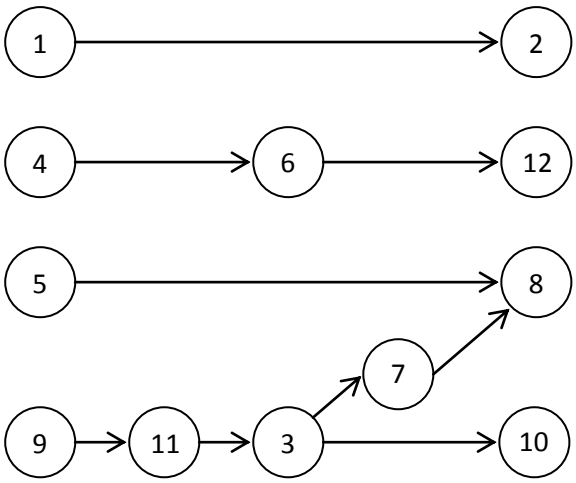


Figure 8.6: Precedence graph for table vice assembly

Table 8.15: Assembly data for table vice assembly

Task	*D*	*T*	*M*
1	+*z*	-	30
2	-*z*	T1	160
3	-*y*	-	50
4	-*y*	-	40
5	+*x*	-	80
6	-*x*	-	50
7	+*y*	-	30
8	+*y*	T1	220
9	-*x*	-	40
10	+*x*	T2	120
11	+*x*	T3	160
12	+*x*	T3	160

Finally, the assembly sequence is then evaluated according to the predetermined objectives. As an example, a feasible assembly sequence, *F* [5, 9, 1, 11, 2, 4, 3, 7, 8, 6, 12, 10] is considered. In this problem, the maximum allowable cycle time (ct_{max}) is 420 seconds. The ct_{max} is calculated from the product demand. As an example, assume the demand for this product is 1500 units per month. Meanwhile, the total working hours per month is 176 hours (22 days/month x 8 hours per day). Therefore, the ct_{max} is calculated as follows:

$$ct_{max} = \frac{\text{Total working hours per month}}{\text{demand per month}}$$

Eq. 8.5

$$ct_{max} = \frac{176 \times 60 \times 60 \text{ seconds}}{1500 \text{ units}} \approx 420 \text{ seconds per unit}$$

This constant will be used during assembly task assignment, where the processing time for each workstation cannot exceed the ct_{max}.

Next, the assembly task is assigned into workstations. The total assembly time for each workstation must not exceed ct_{max}, otherwise the particular assembly

task is assigned to the next workstation. For example, the total assembly time for the first four assembly tasks, 5, 9, 1 and 11 is 310 time unit. Once the fifth assembly task 2 is added together, the total assembly time will be 590 seconds, which is higher than ct_{max}. Therefore, task 2 is assigned into the next workstation.

Table 8.16: Example of assembly task assignment

	ws_1				ws_2				ws_3		ws_4	
F	5	9	1	11	2	4	3	7	8	6	12	10
M	80	40	30	160	160	40	50	30	220	50	160	120
D	*+x*	*-x*	*+z*	*+x*	*-z*	*-y*	*-y*	*+y*	*+y*	*-x*	*+x*	*+x*
T	-	-	-	T3	T1	-	-	-	T1	-	T3	T2
pt				310				280		270		280

After all assembly tasks were assigned, four workstations were established. The maximum processing time, *pt* in all workstations is 310 seconds. Therefore, for this assembly line, the cycle time, *ct* is 310 seconds. From here, the workload variation is calculated using Eq. 6.3.

Workload variation, v = 25 seconds/workstation

The total direction change is 6 and tool change is 4.

Optimisation Results

For optimisation purposes, the MODPSO algorithm is set to run for 1000 iterations and 10 repetitions with different random seeds. The summary of performance indicators for MODPSO and other comparison algorithms for this problem is presented in Table 8.17. The number in brackets shows the weight values assigned based on the performance of a particular indicator. The algorithm with the best performance is assigned weight = 7, the second best weight = 6 and so on.

Table 8.17: Comparison of performance indicators for fixed table vice problem

Algorithm	$\tilde{\eta}$[1]	*ER*[2]	*GD*[2]	*Spacing*[2]	$Spread_{max}$[1]	Weight sum	Rank
MOGA	2 (6)	0.7778 (6)	8.5429 (2)	13.0592 (5)	152.1841 (6)	25	2
ACO	1 (5)	0.9000 (4)	9.2603 (1)	5.7350 (7)	152.1381 (5)	22	5
HGA	2 (6)	0.7778 (6)	8.3207 (4)	13.6870 (3)	152.1315 (4)	23	4
NSGA-II	2 (6)	0.7778 (6)	8.3220 (3)	14.0352 (2)	152.1315 (4)	21	6
MOPSO	1 (5)	0.9091 (3)	7.4286 (5)	13.1759 (4)	145.0896 (3)	20	7
DPSO	1 (5)	0.8571 (5)	2.8437 (6)	12.8492 (6)	131.2974 (2)	24	3
MODPSO	21 (7)	0 (7)	0 (7)	15.9463 (1)	218.1972 (7)	29	1

[1] Larger the better indicator [2] Smaller the better indicator

*Number in brackets are weighting values from the best (weight=7) to worst (weight=1)

From Table 8.17, the proposed MODPSO algorithm shows better performance in all indicators compared to other algorithms, except in *Spacing* indicator. For *Spacing* indicator, the best algorithm is ACO, while the MODPSO algorithm performed worst. The zero values for *ER* and *GD* indicators mean all the non-dominated solutions found by MODPSO algorithm are Pareto optimal solutions. While the $Spread_{max}$ indicator shows that the MODPSO explored the extreme values better than the other algorithms.

The overall performance of algorithms to optimise the fixed table vice is determined by summation of the assigned weight. Based on the total weight, the proposed MODPSO algorithm is ranked in the first place. This is followed by MOGA in the second place, while DPSO is in third place.

The non-dominated solutions for the vice problems acquired using MODPSO are presented in Table 8.18.

Table 8.18: Non-dominated solution of vice problem using MODPSO

No	f_1	f_2	f_3	f_4	f_5	Assembly sequence
1	3	1	240	7	77.14	4-9-6-1-2-12-11-5-3-7-8-10
2	3	2	220	7	57.14	9-11-3-4-1-2-10-5-6-7-8-12
3	3	2	280	6	90	4-1-6-9-11-5-3-7-8-12-10-2
4	3	3	250	6	60	4-6-9-1-2-11-3-10-12-5-7-8
5	3	3	280	5	52	9-5-11-4-3-6-12-10-1-7-8-2
6	4	1	220	7	57.14	4-9-6-1-11-12-2-3-5-7-8-10
7	4	1	280	6	90	9-5-11-3-4-6-7-1-2-12-8-10
8	4	2	220	6	30	9-11-3-4-1-6-7-10-5-8-12-2
9	4	2	280	5	52	9-5-11-3-4-6-7-1-2-8-12-10
10	4	3	370	4	85	4-9-6-5-11-3-1-7-8-10-12-2
11	4	4	310	4	25	1-9-5-11-3-4-6-2-7-8-10-12
12	4	5	250	5	22	4-1-2-6-9-11-3-10-12-5-7-8
13	5	1	240	6	50	1-5-4-9-6-12-11-3-7-10-8-2
14	5	1	320	5	92	5-4-1-6-9-11-12-3-7-2-8-10
15	5	2	320	4	35	5-4-9-6-1-12-11-3-7-8-2-10
16	5	4	240	5	12	1-9-11-4-3-7-10-6-2-12-5-8
17	5	4	290	4	5	4-9-6-11-3-5-12-1-7-8-10-2
18	5	4	420	3	40	4-1-6-9-2-11-12-5-3-7-8-10
19	5	5	410	3	30	9-1-5-11-4-3-7-8-2-6-12-10
20	6	3	260	5	32	4-9-11-3-6-12-1-7-5-10-2-8
21	6	3	390	3	10	4-6-9-5-11-3-7-1-12-10-2-8

The Table 8.18 shows that the MODPSO was able to search for 21 non-dominated solutions with the optimisation set up as explained earlier. The assembly direction change objective (f_1) ranges from 3 to 6 changes, while the assembly tool change objective (f_2) ranges from 1 to 5. Meanwhile the cycle time varies between 220 to ct_{max} (420) resulting in different numbers of workstations. The minimum number of workstations to fulfil the monthly demand is 3 workstations, while the maximum workstation number is 7. On the other hand, the minimum workload variation in the non-dominated solutions is 5 seconds/workstations in solution 17.

From this result, if the ASP and ALB optimisation is performed sequentially, the best solution for ASP objectives (f_1 and f_2) is solution number 1 because both

assembly direction and tool changes are at minimum values and dominate all other solutions. It leaves no other option for the ALB but the solution with 240 seconds of cycle time, 7 workstations and 77.14 seconds/workstation of workload variation. However, when the optimisation for ASP and ALB are integrated, we found that there are better solutions for ALB compared to solution 1, such as solutions 2, 8 and 16. This example clearly shows the benefit of integrating ASP and ALB optimisation.

The non-dominated solution using MODPSO for this problem is plotted in matrix plot as in Figure 8.7. The solution spread can clearly be seen in the diagonal boxes that plotted f_i versus f_i. Based on this figure, the solutions were spread uniformly for objective f_1, f_2 and f_4 with similar gaps between adjacent solutions. On the other hand for f_3, three different gaps are identified within the adjacent non-dominated solutions, which are 10, 20 and 50. From all solutions, only the adjacent solution between 10 and 15 come out with the largest gap. Meanwhile, the adjacent solution gaps for f_5 are varying between 2 and 17.14. Since this objective is calculated from Eq. 6.3 that depends on cycle time and number of workstations, it very difficult to have uniform gaps because when the cycle time is increased, the number of workstations will decrease and vice versa.

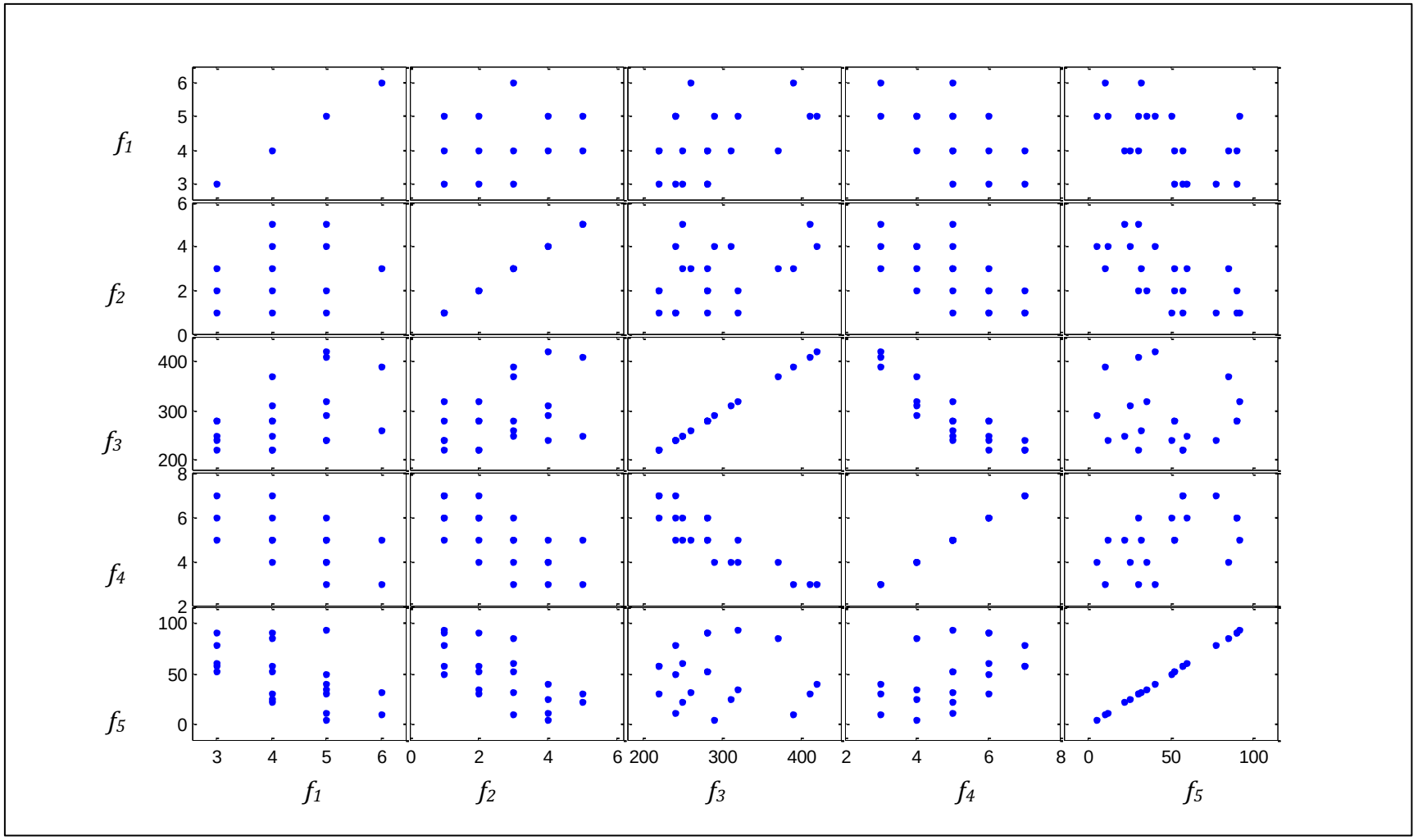


Figure 8.7: Non-dominated solution spread for the table vice problem using MODPSO

8.3.2 Assembly of Toy Train

The second real-world example is assembly of a toy train. The picture of the real product is presented in Figure 8.8.

Figure 8.8: Picture of toy train

The toy train assembly example consists of 39 parts, as presented in Figure 8.9. The liaison matrix that recorded the assembly liaison or task between one part and another part is established as shown in Table 8.19. The liaison matrix comes out with 40 liaisons which explain the task numbers in this assembly example.

Next, the precedence relations between assembly tasks are determined by applying De Fazio's question and answer procedure, the summary from which is presented in Table 8.20. The summary of assembly precedence constraint is presented in precedence diagram (Figure 8.10).

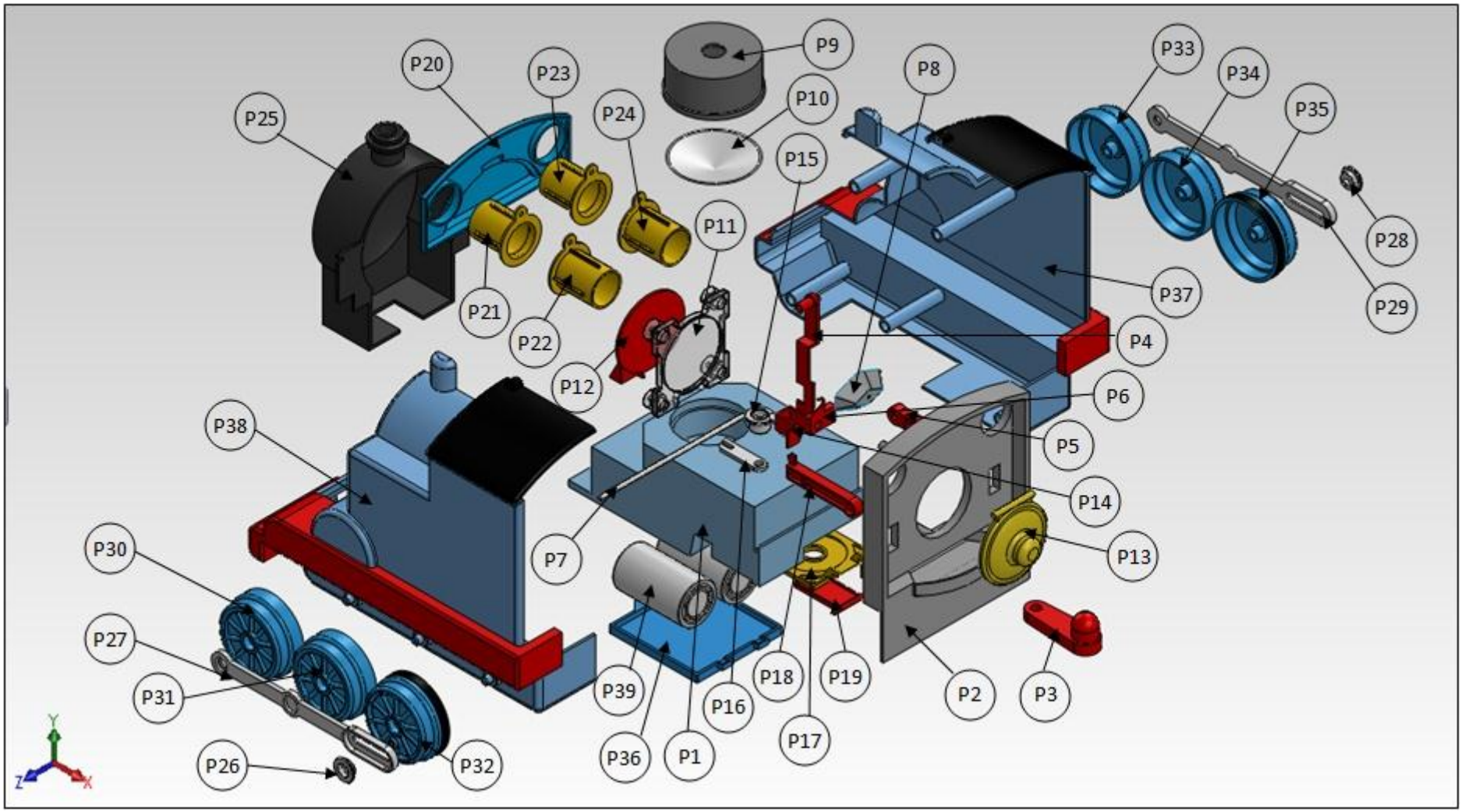


Figure 8.9: Exploded model of toy train assembly

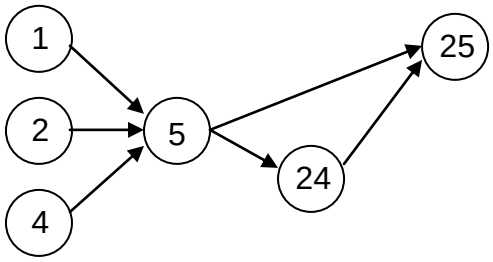
Table 8.19: Liaison matrix for toy train assembly

i \ j	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24	P25	P26	P27	P28	P29	P30	P31	P32	P33	P34	P35	P36	P37	P38	P39
P1	-	5							10	9				2	3	1	22																			39	24		38
P2		-	4		12	14	13	15			7	8	6					11				18		19															
P3			-																																			
P4				-	40														21																			
P5					-																																	
P6						-																																
P7							-																															
P8								-																														
P9									-																													
P10										-																												
P11											-																											
P12												-																										
P13													-																									
P14														-																								
P15															-																							
P16																-																						
P17																	-																					
P18																		-																				
P19																			-																			
P20																				-	20																	
P21																					-	16																
P22																						-																
P23																							-	17														
P24																								-														
P25																									-												23		
P26																										-												28	
P27																											-			27		31							
P28																												-									34		
P29																													-				33		37				
P30																														-								26	
P31																															-							29	
P32																																-						30	
P33																																	-				32		
P34																																		-			35		
P35																																			-		36		
P36																																				-			
P37																																					-	25	
P38																																						-	
P39																																							-

Table 8.20: Summary of De Fazio's Q&A for toy train assembly

Task *i*	Q1 (Precedence)	Q2 (Follower)
1	-	5,24,25
2	-	5,24,25
3	-	24,25
4	-	5,24,25
5	1,2,4	24,25
6	-	-
7	-	10,24,25
8	3,7	24,25
9	-	24,25
10	9	24,25
11	-	13,14,24,25
12	-	13,24,25
13	8,11,12	24,25
14	7,8,11	24,25
15	8,7	24,25
16	-	24,25
17	-	24,25
18	16	24,25
19	17	24,25
20	17, 19	24,25
21	-	24,25
22	-	21,24,25
23	-	24,25
24	1,2,3,4,5,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,40	25
25	1,2,3,4,5,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,40	-
26	-	28,27
27	26	28
28	26,27	-
29	-	27
30	-	-
31	-	-
32	-	33,34
33	-	34
34	32,33	-
35	24	-
36	24	-
37	35	-
38	-	-
39	38	-
40	-	21

The precedence constraint for this problem is established from Table 8.20. The assembly tasks from column Q1 will be the precedence, while the tasks in column Q2 will be the follower for assembly task *i*. For example, if we consider assembly task 5, the precedence for this tasks are tasks 1, 2 and 4, while the follower tasks are 24 and 25. Besides that, task 24 is the precedence for task 25. Therefore, the precedence graph mapping for task 5 will be as follows.



However, according to the transitivity of the precedence constraints, the shortest paths between two generic nodes are removed (Fouda et al., 2001). In the example above, there are two translation paths from task 5 to task 25, either via task 24 or directly to task 25. In this case, the shortest path is eliminated because it is inapplicable until task 24 is completed. The final precedence diagram for the toy train assembly is presented in Figure 8.10.

Next the assembly data which consists of assembly direction (*D*), assembly tool (*T*) and assembly time (*M*) are identified. For task 1 which involves part P1 and P16, part P1 is defined as the fixed part and part P16 as the moving part. Therefore, to perform this task, P16 will be assembled into –*y* direction. In this task, no assembly tool is required and the recorded assembly time is 3 seconds. The rest of the assembly data for the toy train assembly is presented in Table 8.21.

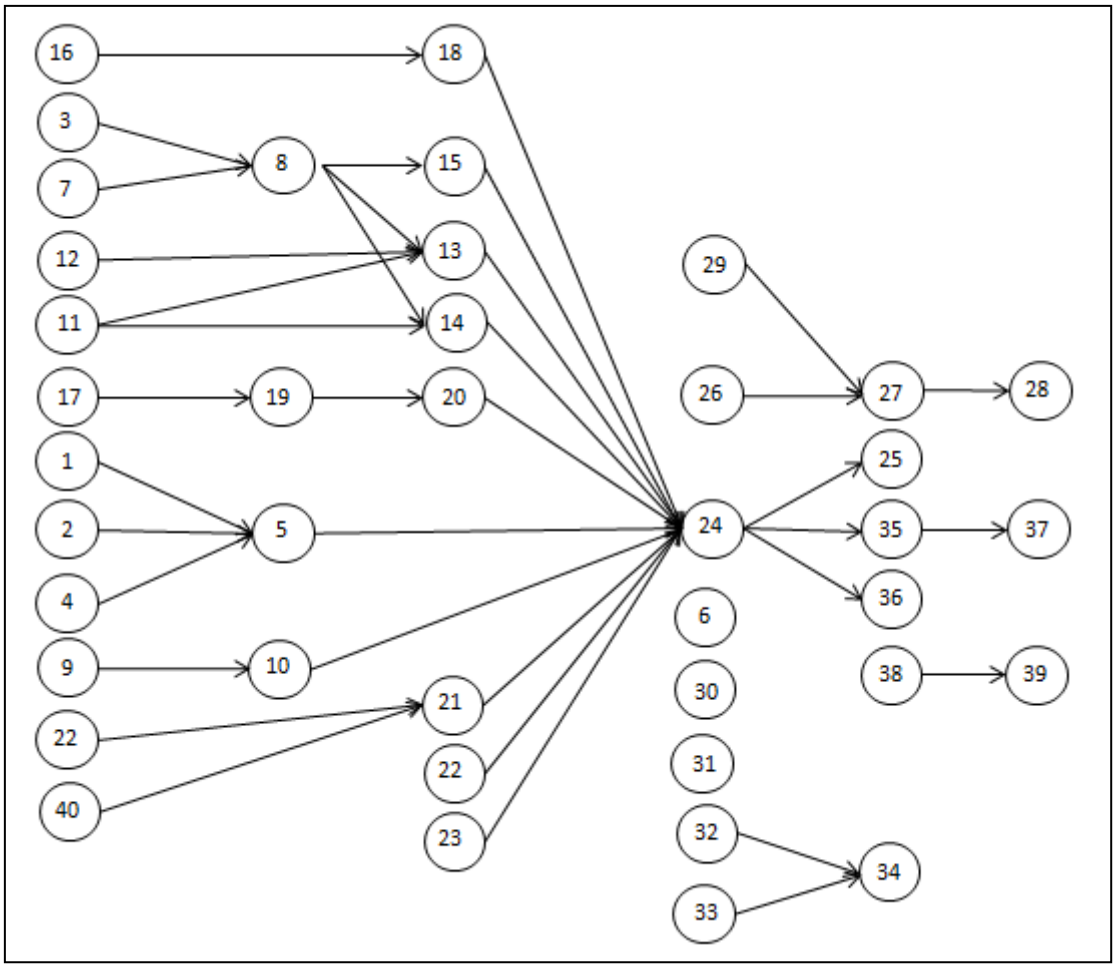


Figure 8.10: Precedence graph of toy train assembly

Table 8.21: Assembly data for toy train assembly

Task	*D*	*T*	*M*	Task	*D*	*T*	*M*	Task	*D*	*T*	*M*	Task	*D*	*T*	*M*
1	*-y*	-	3	11	*x*	-	8	21	*-z*	-	9	31	*-z*	T4	19
2	*-y*	-	3	12	*x*	-	5	22	*y*	T1	62	32	*z*	-	3
3	*-y*	T4	17	13	*-z*	-	7	23	*-z*	-	6	33	*z*	-	3
4	*-x*	-	2	14	*x*	T1	38	24	*z*	T5	52	34	*z*	T6	17
5	*-y*	T2	64	15	*x*	T3	24	25	*-z*	-	132	35	*z*	T4	24
6	*-x*	-	7	16	*x*	-	2	26	*-z*	-	4	36	*z*	T4	24
7	*x*	T1	59	17	*x*	-	2	27	*-z*	-	3	37	*z*	T4	18
8	*x*	T2	32	18	*x*	T1	15	28	*-z*	T6	17	38	*y*	-	10
9	*-y*	-	3	19	*x*	T1	15	29	*-z*	T4	24	39	*y*	T2	16
10	*-y*	-	6	20	*x*	-	2	30	*-z*	T4	24	40	*-z*	T4	12

Optimisation Results

The optimisation for assembly of the toy train is performed with 1000 runs and 10 repetitions of different random seeds. The summary of algorithms' performance in optimising this problem is presented by performance indicators as shown in Table 8.22.

Table 8.22: Comparison of performance indicators for toy train assembly

Algorithm	$\tilde{\eta}$[1]	ER[2]	GD[2]	$Spacing$[2]	$Spread_{max}$[1]	Weight sum	Rank
MOGA	1 (4)	0.9333 (5)	2.7579 (4)	2.0711 (2)	37.1462 (4)	19	4
ACO	0 (3)	1.0000 (4)	3.6281 (3)	1.1662 (6)	34.4771 (3)	19	4
HGA	7 (6)	0.6667 (6)	2.3727 (6)	1.3965 (5)	38.9605 (5)	28	2
NSGA-II	5 (5)	0.3750 (7)	1.0691 (7)	5.1956 (1)	34.2004 (1)	21	3
MOPSO	0 (3)	1.0000 (4)	6.3305 (1)	1.1130 (7)	35.6733 (2)	17	7
DPSO	0 (3)	1.0000 (4)	5.5870 (2)	1.9213 (3)	39.4705 (6)	18	6
MODPSO	11 (7)	0.6071 (6)	2.4262 (5)	1.4079 (4)	41.9633 (7)	29	1

[1] Larger the better indicator [2] Smaller the better indicator

*Number in brackets are weighting values from the best (weight=7) to worst (weight=1)

Based on Table 8.22, the MODPSO algorithm performed better in finding more Pareto optimal solutions presented by $\tilde{\eta}$ indicator. This is followed by HGA and NSGA-II with 7 and 5 Pareto solutions respectively. Meanwhile, the NSGA-II shows better performance in *ER* and *GD* indicators. For *ER* and *GD* indicators, the MODPSO algorithm placed in second and third rank correspondingly. On the other hand, in terms of solution uniformity represented by the *Spacing* indicator, the best algorithm is the MOPSO algorithm. For this indicator, the MODPSO algorithm is only in fourth place, while the NSGA-II is in last place. The MODPSO algorithm once again performed better compared to other algorithms in $Spread_{max}$ indicator. It shows that the MODPSO algorithm has better ability to explore the extreme solutions.

The overall algorithm performance shown by the summation of weight indicates that the MODPSO algorithm leads the board with only small differences compared to HGA. Meanwhile, the NSGA-II algorithm is in third rank although it

shows good performance in $\tilde{\eta}$, *ER* and *GD* indicators. But in terms of solution uniformity and extreme solution exploration, the NSGA-II did not perform well.

The non-dominated solution that was acquired from the assembly of the toy train is presented in Table 8.23. The MODPSO algorithm able to find 28 non-dominated solutions with different assembly sequences.

Table 8.23: Non-dominated solutions of toy train assembly

No.	f_1	f_2	f_3	f_4	f_5	No.	f_1	f_2	f_3	f_4	f_5
1	7	17	132	8	32.88	**15**	17	18	134	6	1.833
2	8	14	132	7	18.71	**16**	17	21	133	6	0.833
3	9	13	132	7	18.71	**17**	18	10	132	7	18.71
4	10	12	134	7	20.71	**18**	18	17	135	6	2.833
5	14	11	133	8	33.88	**19**	19	7	135	8	35.88
6	14	12	132	8	32.88	**20**	19	9	132	7	18.71
7	15	11	132	8	32.88	**21**	19	20	133	6	0.833
8	15	20	135	6	2.833	**22**	20	7	132	8	32.88
9	15	21	134	6	1.833	**23**	20	13	135	6	2.833
10	16	10	132	8	32.88	**24**	20	16	134	6	1.833
11	16	12	132	7	18.71	**25**	20	18	133	6	0.833
12	16	18	135	6	2.833	**26**	22	15	134	6	1.833
13	17	8	132	8	32.88	**27**	24	8	132	7	18.71
14	17	11	133	7	19.71	**28**	25	13	134	6	1.833

The objective value for f_1 ranges between 7 and 25, while the f_2 spread is between 7 and 21. However, there is no single solution that comes out with minimum value for both f_1 and f_2 as found in the vice assembly case study. The minimum f_1 sequence can be found in solution 1, while the minimum f_2 are in solutions 19 and 22. For the cycle time (f_3), the minimum and maximum cycle times are 132 and 135 respectively. These values are as predicted from the maximum task time (task 25) and given maximum cycle time (ct_{max}) defined earlier.

Meanwhile, the number of workstations (f_4) presents three options in the results; the assembly line with 6, 7 or 8 workstations. Finally the workload variation (f_5)

presents the minimum value of 0.83 seconds per workstation in solution 16, 21 and 25. This value means the average idle time in each workstation is only 0.83 seconds for 133 seconds of cycle time. It also means that for the minimum f_5, the average of busy time for worker is 99.4%, which is almost perfectly balanced.

Figure 8.11 shows the matrix plot of non-dominated solution for the MODPSO algorithm. The solution spread for each objective can be observed on the diagonal graph on which is plotted the f_i versus f_i. The first objective, f_1 was spread nearly uniformly with gap 1 and 2 except between non-dominated solutions 4 to 5 that have gap 4. The f_2 was also spread nearly uniformly between minimum and maximum with gap 1 and 2. The f_3 and f_4 objectives were spread uniformly with gap 1 between minimum and maximum values. Finally in f_5, it is found that the solutions were spread within three clusters. In all clusters, the nearly uniform gap is identified. These clusters are formed because of the discrete changing in number of workstations in Eq. 6.3.

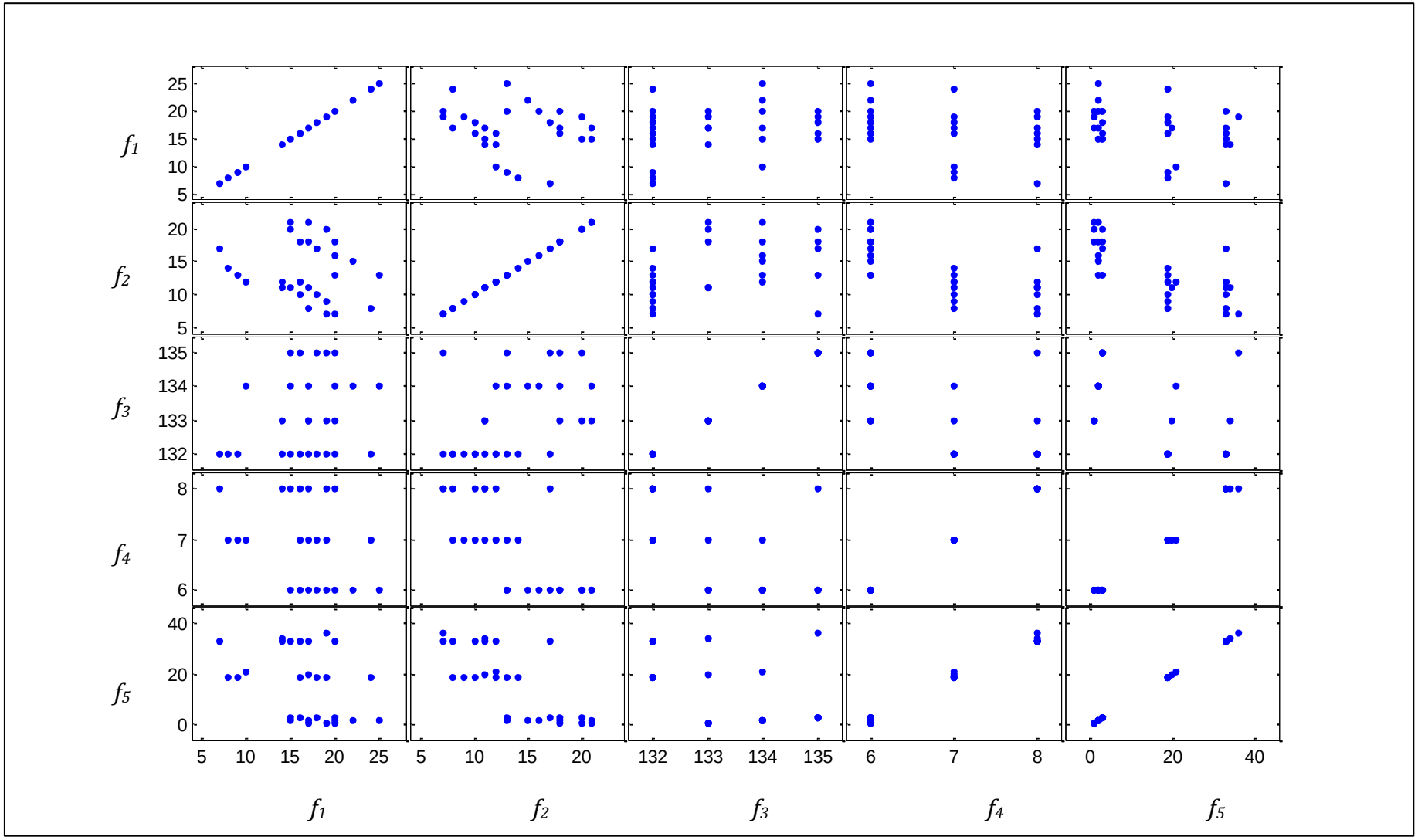


Figure 8.11: Non-dominated solution spread for toy train assembly using MODPSO

8.3.3 Assembly of Nissan Pathfinder Engine (Bautista and Pereira, 2007)

This is a real-life problem that was adopted from Bautista and Pereira, (2007). The presented problem is the assembly of the Nissan Pathfinder motor engine that consists of 140 tasks, as shown in Figure 8.12. In the original article, Bautista and Pereira consider two types of line balancing problems: Time and Spaced constraint of ALB Problem (TSALBP) and Simple Assembly Line Balancing Problem type-I (SALBP-I). The SALBP-I minimise the number of workstation for given a fixed value of cycle time. In this work, we will consider the SALBP-I because this type of problem is highly related and comparable with our works on integrated ASP and ALB. Since the original article only considers the ALB problem, the ASP data (i.e. assembly direction and tool) will be generated using the test problem generator that was developed in Chapter 5.

The objective for line balancing in the original work is to minimise the number of workstations for given maximum cycle time. In this case, the allowable maximum cycle time is 180 seconds. The assembly data for this problem is presented in Table 8.24. From Table 8.24, the assembly time (*M*) is adopted from Bautista's original article, whilst the assembly direction (*D*) and assembly tool (*T*) information is generated using a test problem generator.

Since the original article only presented one best solution based on SALBP-I, the Ant Colony Optimisation (ACO) algorithm as used in Bautista and Pereira (2007) was replicated for comparison purposes.

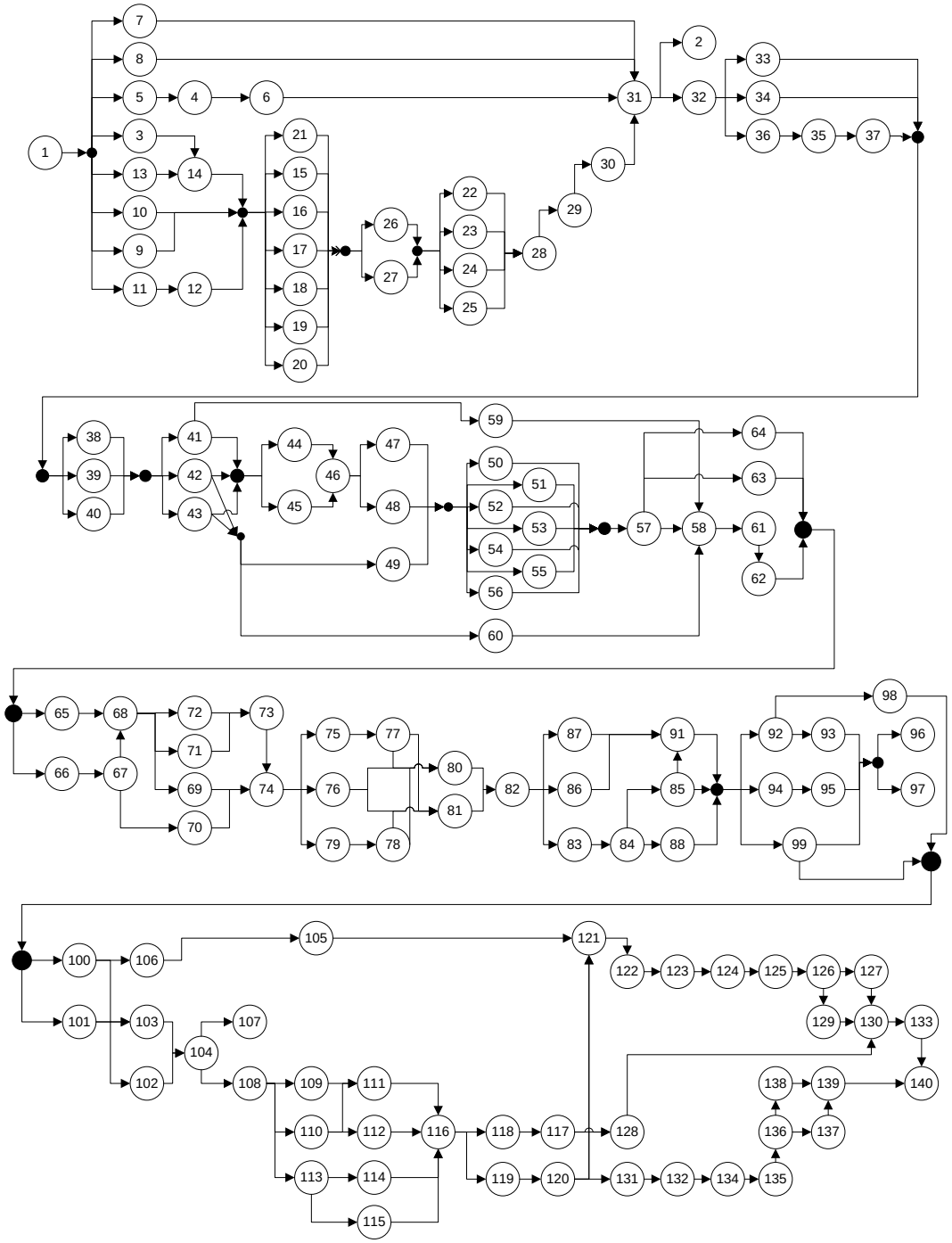


Figure 8.12: Precedence diagram of the 140 task problem (Bautista and Pereira, 2007)

Table 8.24: Nissan Pathfinder assembly information

i	*D*	*T*	*M*	*i*	*D*	*T*	*M*	*i*	*D*	*T*	*M*	*i*	*D*	*T*	*M*
1	+*x*	3	60	36	-*z*	20	25	71	-*z*	26	10	106	-*y*	6	25
2	-*y*	27	75	37	+*y*	16	15	72	+*x*	14	10	107	+*y*	17	5
3	+*x*	17	20	38	+*x*	3	5	73	+*z*	33	40	108	-*y*	7	5
4	+*x*	19	60	39	-*x*	28	5	74	+*y*	25	25	109	+*x*	26	5
5	-*x*	18	20	40	-*y*	12	5	75	+*x*	14	10	110	-*x*	8	5
6	+*x*	33	60	41	-*y*	24	60	76	-*y*	27	10	111	+*z*	27	10
7	-*y*	24	45	42	-*y*	4	15	77	+*y*	6	15	112	-*x*	15	10
8	+*y*	12	10	43	-*y*	4	15	78	-*z*	26	15	113	+*z*	7	15
9	-*y*	35	20	44	-*z*	10	25	79	-*z*	17	15	114	+*z*	13	20
10	-*y*	21	30	45	-*z*	25	25	80	-*x*	30	10	115	+*z*	23	20
11	-*y*	11	15	46	-*z*	28	5	81	-*y*	9	10	116	-*x*	25	45
12	+*x*	29	15	47	-*z*	16	35	82	+*z*	28	10	117	-*y*	19	20
13	+*x*	8	15	48	+*y*	35	35	83	-*y*	36	20	118	-*y*	26	25
14	+*y*	22	10	49	+*y*	15	5	84	-*x*	8	10	119	+*y*	36	25
15	-*y*	22	8	50	-*z*	1	15	85	-*z*	33	20	120	-*x*	32	20
16	-*y*	19	8	51	+*y*	15	25	86	-*z*	7	25	121	+*x*	11	35
17	+*y*	20	80	52	-*x*	18	30	87	+*z*	13	20	122	-*z*	28	15
18	-*z*	28	40	53	+*z*	29	15	88	+*z*	12	15	123	-*z*	31	10
19	+*z*	9	5	54	-*y*	13	15	89	-*z*	5	20	124	+*z*	18	10
20	-*y*	29	5	55	-*y*	1	20	90	+*y*	31	30	125	-*x*	35	20
21	+*y*	18	5	56	-*z*	34	10	91	-*y*	18	20	126	+*x*	7	30
22	-*x*	30	7	57	-*z*	18	10	92	+*y*	20	25	127	-*y*	5	10
23	+*x*	30	7	58	+*x*	11	20	93	+*y*	36	10	128	+*y*	29	25
24	+*y*	6	30	59	-*y*	21	5	94	-*y*	10	5	129	-*y*	36	30
25	-*x*	19	30	60	+*x*	12	20	95	+*y*	2	20	130	+*z*	1	30
26	+*x*	28	5	61	-*x*	15	45	96	-*y*	5	10	131	+*x*	32	40
27	-*z*	4	5	62	-*y*	32	30	97	+*y*	14	5	132	-*x*	2	25
28	+*x*	17	30	63	+*y*	2	30	98	-*z*	9	80	133	-*x*	31	25
29	-*y*	13	10	64	-*z*	23	10	99	-*z*	31	25	134	-*x*	6	20
30	-*x*	35	15	65	+*z*	12	5	100	-*y*	34	10	135	-*x*	8	15
31	-*x*	24	10	66	-*z*	3	10	101	-*z*	18	10	136	-*y*	34	20
32	+*x*	3	15	67	-*z*	34	15	102	-*z*	10	20	137	-*y*	28	30
33	+*z*	27	30	68	-*y*	22	60	103	+*y*	23	30	138	-*x*	21	30
34	-*y*	9	10	69	-*z*	18	10	104	-*x*	33	5	139	-*z*	5	15
35	+*x*	13	5	70	-*x*	16	30	105	-*z*	15	30	140	+*y*	33	120

i: Assembly task;*T*: Assembly tool; *D*: Assembly direction; *M*: Assembly time

Optimisation Results

The experiment for each algorithm is run with 1000 iterations. The summary of performance indicators for this problem is presented in Table 8.25.

Table 8.25: Comparison of performance indicators for Nissan engine assembly

Algorithm	$\tilde{\eta}$[1]	*ER*[2]	*GD*[2]	*Spacing*[2]	$Spread_{max}$[1]	Weight sum	Rank
MOGA	0 (1)	1.0000 (1)	7.1053 (2)	3.2703 (5)	59.1633 (1)	10	7
ACO	9 (5)	0.5909 (6)	2.9969 (6)	5.7892 (1)	67.1309 (6)	24	3
HGA	1 (2)	0.9787 (2)	7.2084 (1)	2.5239 (6)	61.4029 (4)	15	6
NSGA-II	16 (6)	0.1111 (7)	0.5708 (7)	5.6542 (2)	63.6324 (5)	27	1
MOPSO	2 (3)	0.9583 (3)	6.4129 (3)	2.4829 (7)	59.6939 (3)	19	4
DPSO	3 (4)	0.9143 (4)	5.5780 (4)	3.7260 (4)	59.5544 (2)	18	5
MODPSO	18 (7)	0.6170 (5)	4.1366 (5)	3.9168 (3)	68.0843 (7)	27	1

[1] Larger the better indicator [2] Smaller the better indicator

*Number in brackets are weighting values from the best (weight=7) to worst (weight=1)

The number of non-dominated solutions in Pareto optimal, $\tilde{\eta}$ indicates that the MODPSO algorithm performed better compared to other algorithms. This is closely followed by NSGA-II in second place, differing only in two numbers. For this indicator, the ACO algorithm that was used in the original article was able to find 9 Pareto optimal from 22 non-dominated solutions. In the meantime, the NSGA-II consistently performed better in *ER* and *GD* indicators as found in medium size problems (i.e. 45 tasks and toy train problems). For both performance indicators, the ACO is in second position, while the proposed MODPSO is in third position. Meanwhile, the *Spacing* indicator is led by MOPSO algorithm, while the MODPSO, NSGA-II and ACO algorithms are among in the lowest positions. In the $Spread_{max}$ indicator, the MODPSO algorithm performed best, and followed by ACO and NSGA-II.

The overall algorithm performance based on the summation of weight values indicates that two algorithms share the leader board, these being MODPSO and NSGA-II. Both algorithms came out with similar weight summation. Meanwhile the ACO algorithm used in the original article ranks third. Although the

MODPSO and NSGA-II share better overall performance, the MODPSO has greater ability to find Pareto optimal solution and explore the extreme solutions. On the other hand, the NSGA-II has advantages in solution accuracy towards Pareto optimal.

In total, the MODPSO algorithm is able to find 47 non-dominated solutions for the engine assembly problem. The non-dominated solution from MODPSO is plotted in Figure 8.13. From this figure, the f_1 is spread evenly in the gaps between 1 and 2. The objective f_2 is evenly spread with gap 1, except solution 3 which is isolated from other solutions. Meanwhile, 80% of the f_3 gaps between adjacent non-dominated solutions vary between 1 and 5, while the gap of the remaining 20% is 9 units. The f_4 axis shows uniform spread which means that the algorithm was able to explore all the workstations between 17 and 28. Finally, the f_5 objective is spread with the gaps ranging from 0.42 to 3. This is because of discrete changing in the number of workstations.

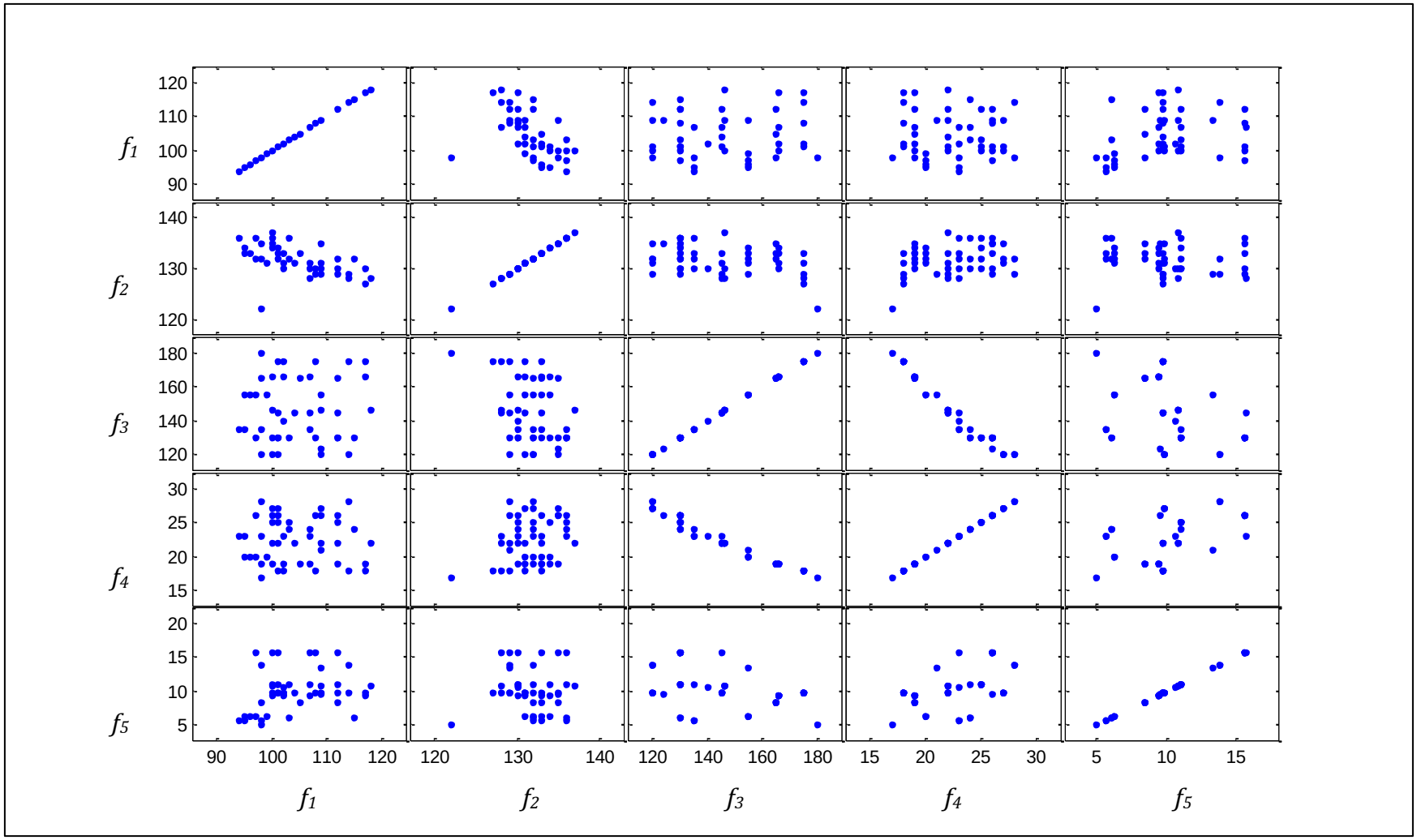


Figure 8.13: Non-dominated solution spread for Nissan Pathfinder engine assembly using MODPSO

Since this problem is adopted from the literature, the result from MODPSO will be compared to the ACO algorithm used in the original article (Bautista and Pereira, 2007). The Bautista's original work, which only considered the ALB problem, presents the best solution with 17 workstations and 180 seconds of cycle time. This result is fully achieved in one non-dominated solution using the ACO algorithm as used in the original article (Bautista and Pereira, 2007). The assembly task assignment for this solution is presented in Table 8.26. In this solution, the cycle time (maximum processing time among all workstations) is 180 seconds, which results in 85 seconds of total idle time and 5 seconds per workstation of workload variation.

Table 8.26: The best line balancing result from (Bautista and Pereira, 2007)

Station, *i*	Assembly sequence	Assembly time (pt_i)	Idle time (ct-pt_i)
1	[1 7 9 10 13 14]	180	0
2	[3 11 15 16 17 18 19]	176	4
3	[4 5 6 8 20 21 22 26 27]	177	3
4	[23 24 25 28 29 30 31 32 33]	177	3
5	[2 34 35 36 37 38 39 40 42 43 49]	180	0
6	[41 44 45 46 47 59 60]	175	5
7	[48 50 51 52 53 54 55 56 57]	175	5
8	[12 58 61 62 63 64 65 66 67]	180	0
9	[68 69 70 71 72 73]	160	20
10	[74 75 76 77 78 79 80 81 82 83 84 86]	175	5
11	[85 87 88 89 90 91 92 99]	175	5
12	[98 100 101 102 103 104 106]	180	0
13	[93 94 105 108 109 110 111 112 113 114	180	0
14	[117 118 119 120 128 131 132]	180	0
15	[95 96 97 107 134 135 136 137 138 139]	170	10
16	[121 122 123 124 125 126 127 129]	160	20
17	[130 133 140]	175	5
		Total idle time	85

For the same problem, the MODPSO is also able to assign assembly tasks into 17 workstations. However the MODPSO algorithm came out with 178 seconds

of cycle time, which is slightly better than the best result presented in the original article. The assembly task assignment for this solution is presented in Table 8.27. The total idle time for this solution is 51 seconds. The workload variation for this solution is 3 seconds per workstation.

Table 8.27: The best line balancing result from MODPSO

Station, i	Assembly sequence	Assembly time, pt_i	Idle time (ct-pt_i)
1	[1 9 10 13 3 14 11 15]	178	0
2	[7 16 17 18 19]	178	0
3	[5 4 6 8 20 21 26 27 22]	177	1
4	[23 24 25 28 29 30 31 32 33]	177	1
5	[2 34 36 35 37 38 39 40 42 43]	175	3
6	[49 41 44 45 46 47 60]	175	3
7	[59 48 50 51 52 53 54 55 12]	175	3
8	[56 57 58 61 62 63 64 65 66]	170	8
9	[67 68 69 70 71 72 73]	175	3
10	[74 75 76 77 79 78 80 81 82 83 84 86]	175	3
11	[85 87 88 89 90 91 92 99]	175	3
12	[98 100 101 102 103 106]	175	3
13	[104 94 93 95 96 97 105 108 109	175	3
14	[116 118 117 119 120 131]	175	3
15	[128 132 107 134 135 136 137	170	8
16	[121 122 123 124 125 126 139 129	175	3
17	[130 133 140]	175	3
		Total idle	51

8.3.4 Assembly of Mixed-Model Table Vice

This is the real life problem of mixed-model assembly for the table vice. In this problem, three different table vice models are assembled in an assembly line. The first model is a fixed table vice which is intended to be securely bolted on the working table. The second model is a portable table vice that can easily be

clamped and removed from any working table. Finally the third model is an angle table vice, which is used to clamp the work-piece at an adjustable angle.

Model A: Fixed Table Vice

The fixed table vice model (Model A) is shown in Figure 8.14. This is a similar model as used in the first real life problem. The liaison matrix and precedence diagram for this model is presented in Table 8.28 and Figure 8.15.

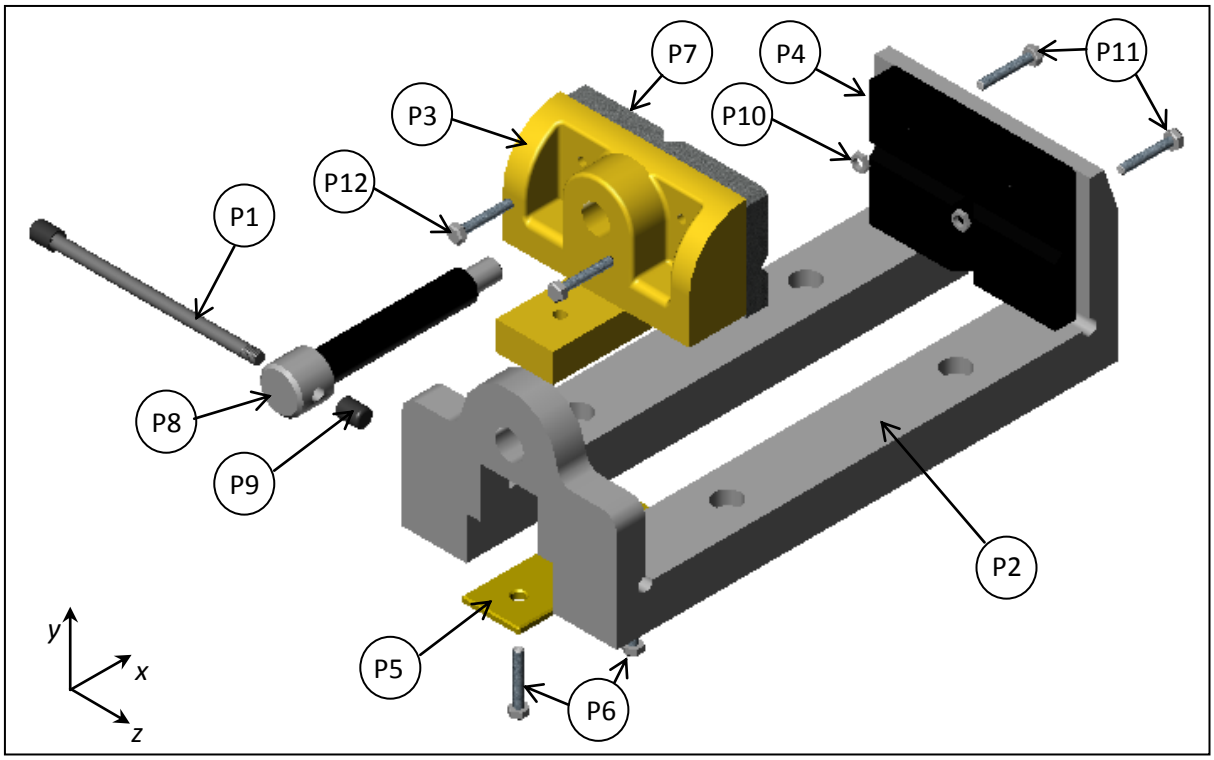


Figure 8.14: Model A-The fixed table vice

Table 8.28: Liaison matrix for model A

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
P1	-							1	2			
P2		-	3	4				5			6	
P3			-		7	8	9	10				
P4				-								
P5					-							
P6						-						
P7							-					11
P8								-				
P9									-			
P10										-	12	
P11											-	
P12												-

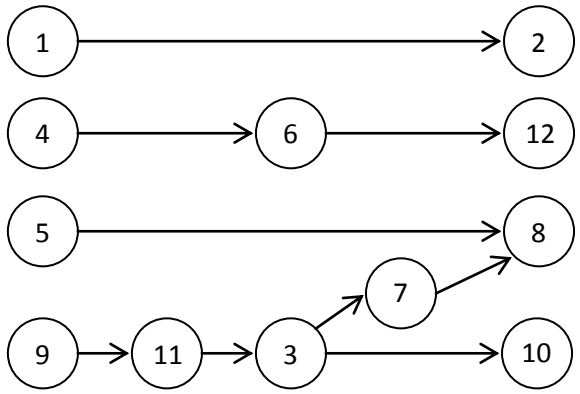


Figure 8.15: Precedence diagram for model A

Model B: Portable Table Vice

The portable table vice consists of 17 parts, as shown in Figure 8.16. In contrast to the fixed table vice, it uses a different vice body (P2) which is assembled to the table using a clamping mechanism. The liaison matrix for this model is presented in Table 8.29. This table shows that the model has 17 assembly tasks. The precedence diagram for this model is presented in Figure 8.16.

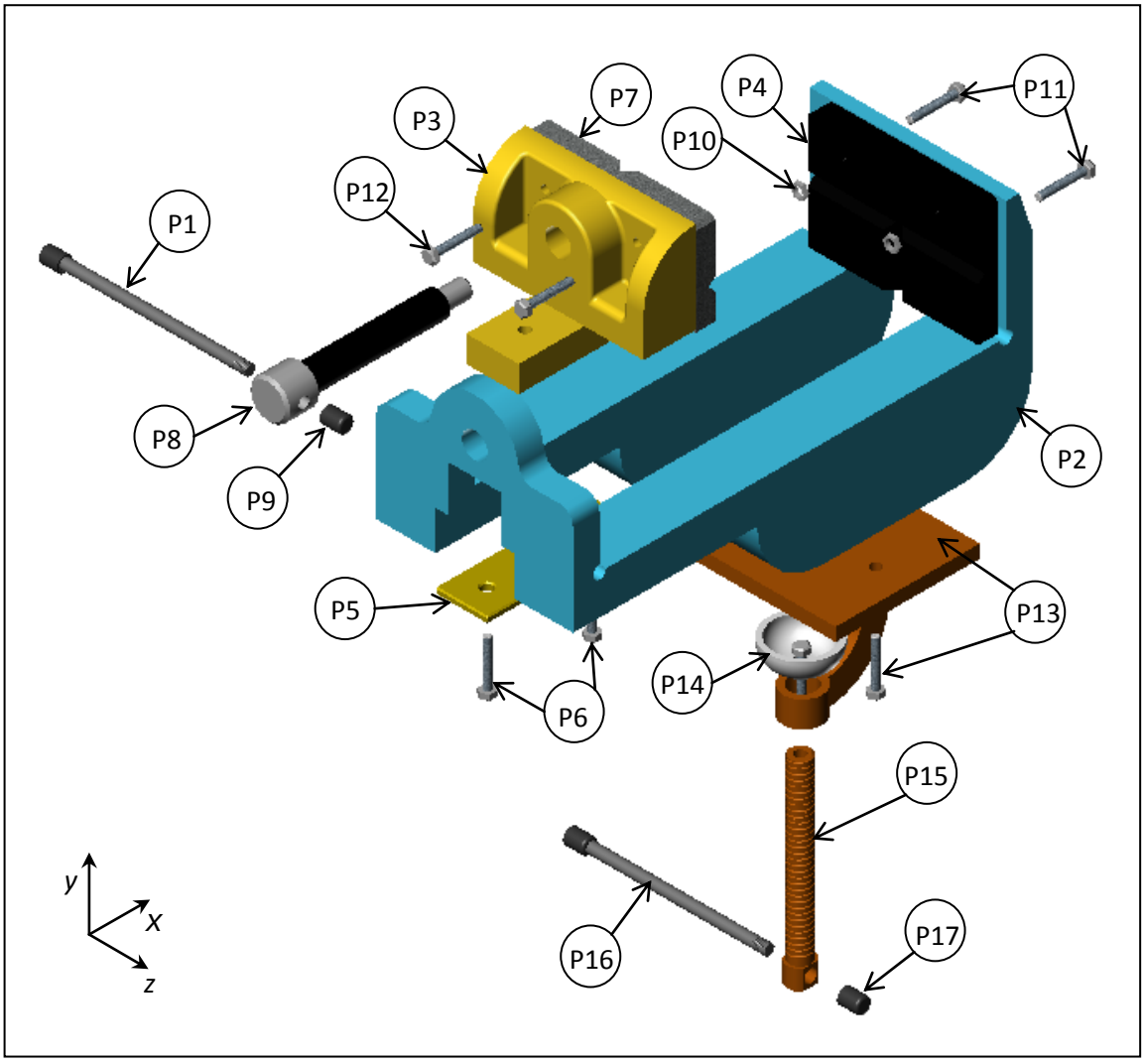


Figure 8.16: Model B – Portable table vice

Table 8.29: Liaison matrix for model B

	P 1	P 2	P 3	P 4	P 5	P 6	P 7	P 8	P 9	P 10	P 11	P 12	P 13	P 14	P 15	P 16	P 17
P1	-							1	2								
P2		-	3	4				5			6		13				
P3			-		7	8	9	10									
P4				-													
P5					-												
P6						-											
P7							-					11					
P8								-									
P9									-								
P10										-	12						
P11											-						
P12												-					
P13													-		14		
P14														-	15		
P15															-	16	
P16																-	17
P17																	-

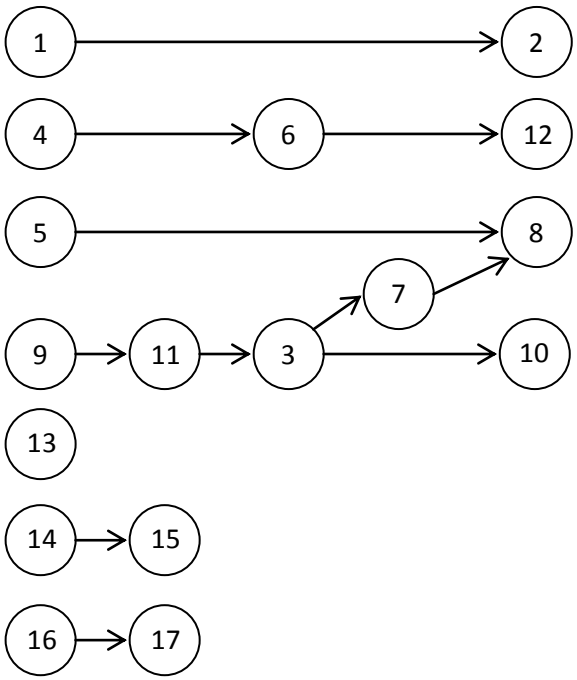


Figure 8.17: Precedence diagram for model B

Model C: Angle Table Vice

The angle table vice, as shown in Figure 8.18, allows work-piece clamping at different angles. It consists of 21 parts, where part P21 is bolted onto the working table and P13 and P17 allow the vice body (P2) to be adjusted to the required angle. The liaison matrix for this model is presented in Table 8.30, whilst the precedence matrix is in Figure 8.19. This assembly problem consists of 19 tasks, as shown in Table 8.30. In this case, similar assembly task numbers may represent different assembly relations between parts. For example, assembly task 13 in Table 8.30 represents assembly relations between parts P2 and P15, while assembly task 13 for model B (Table 8.29) represents assembly relation between parts P2 and P13.

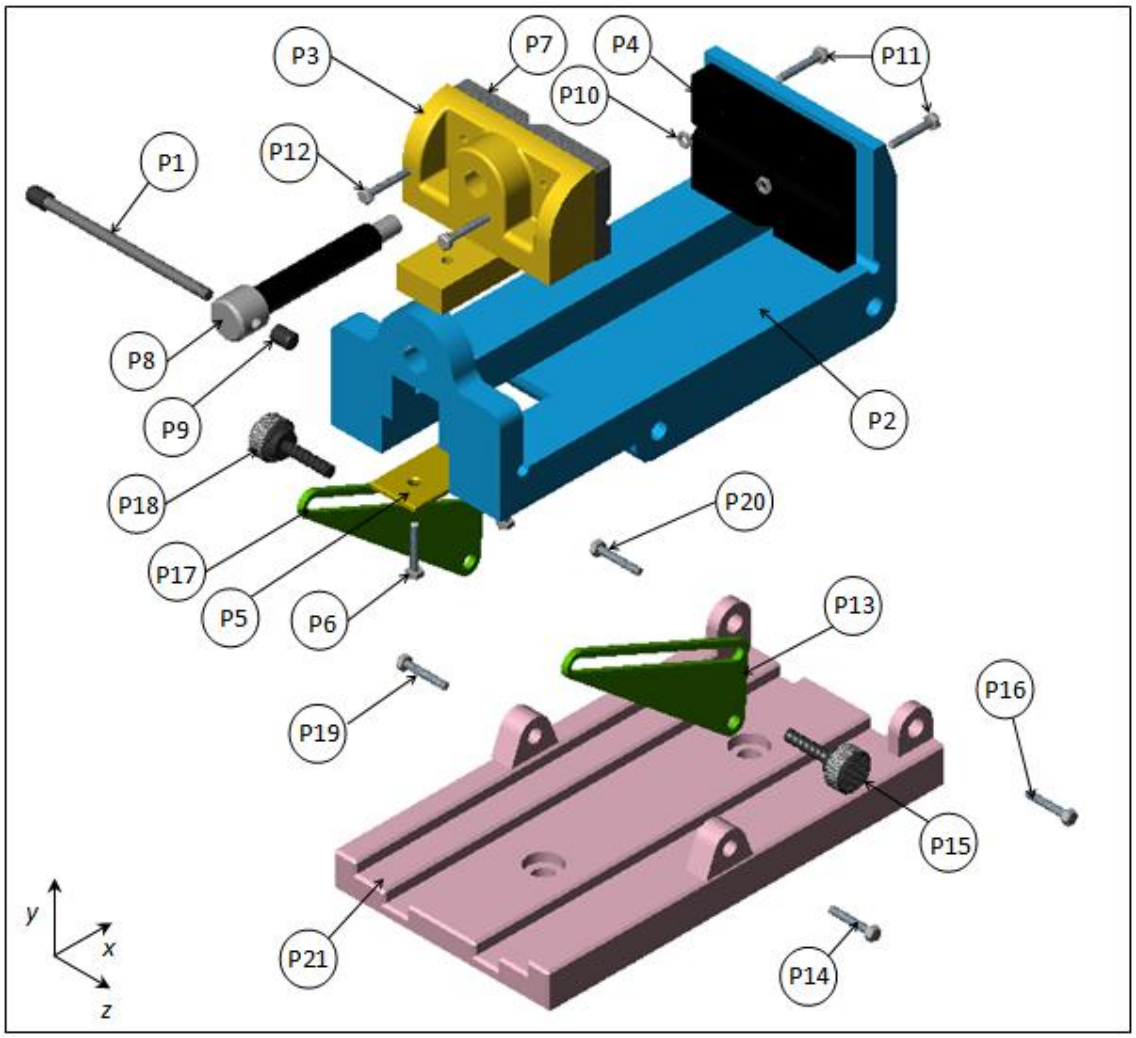


Figure 8.18: Model C – Angle table vice

Table 8.30: Liaison matrix for model C

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21
P1	-							1	2												
P2		-	3	4				5			6				13	18		14		19	17
P3			-		7	8	9	10													
P4				-																	
P5					-																
P6						-															
P7							-					11									
P8								-													
P9									-												
P10										-	12										
P11											-										
P12												-									
P13													-	15							
P14														-							
P15															-						
P16																-					
P17																	-		16		
P18																		-			
P19																			-		
P20																				-	
P21																					-

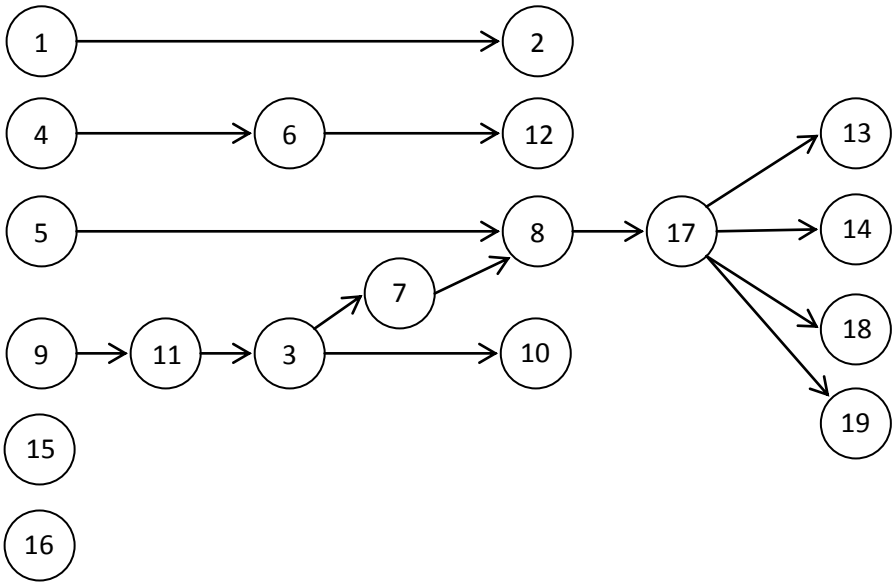


Figure 8.19: Precedence diagram for model C

Mixed-Model Formulation

In order to formulate the integrated mixed-model ASP and ALB problem for models A, B and C, a joint precedence approach is used in this work. The joint precedence diagram represents the precedence constraints for all models in one diagram. To establish the joint precedence diagram, the followers for specific tasks in each model are bundled together in one graph. As an example, the follower for task 14 in joint diagram is task 15, adopted from model B although a similar task in model C did not have any follower. The complete joint precedence diagram for this problem is presented in Figure 8.20.

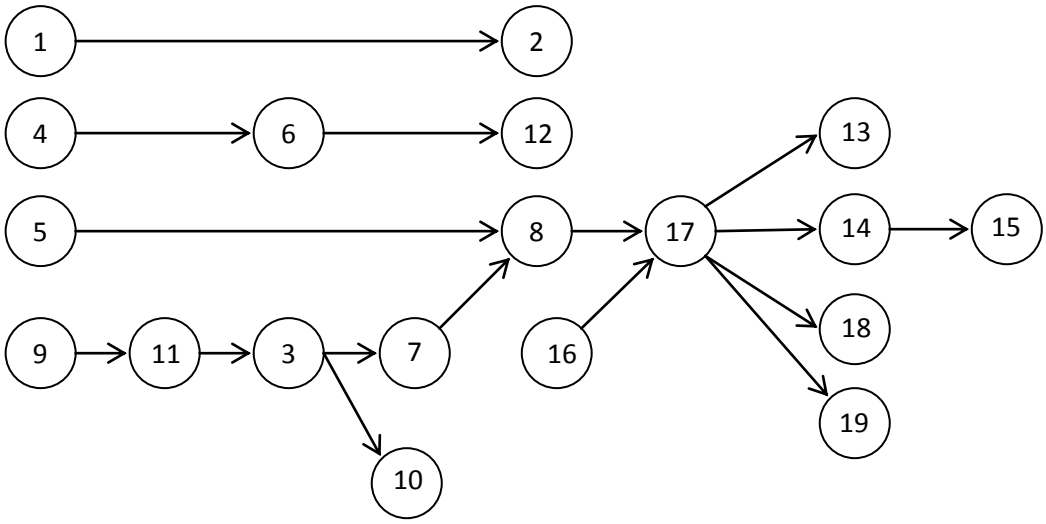


Figure 8.20: Joint precedence diagram for table vice assembly

The assembly data for all models is presented in Table 8.31. The assembly data consists of assembly direction (*D*), assembly tool (*T*) and assembly time (*M*). For example in task 1, the assembly direction is in *z* direction, no assembly tool is involved and 30 seconds of assembly time is required. The value 0 in this table indicates that the particular task is not applicable for the specific model. For example, the assembly tasks from 13 to 19 are inapplicable for model A, since this model only has 12 tasks. The maximum cycle time for model A, B and C are 420, 450 and 470 seconds respectively.

Table 8.31: Assembly data for mixed-model table vice assembly

Task	Model A			Model B			Model C		
	D	*T*	*M*	*D*	*T*	*M*	*D*	*T*	*M*
1	*z*	-	30	*z*	-	30	*z*	-	30
2	*-z*	2	160	*-z*	2	160	*-z*	2	160
3	*-y*	-	50	*-y*	-	50	*-y*	-	50
4	*-y*	-	40	*-y*	-	40	*-y*	-	40
5	*x*	-	80	*x*	-	80	*x*	-	80
6	*-x*	-	50	*-x*	-	50	*-x*	-	50
7	*y*	-	30	*y*	-	30	*y*	-	30
8	*y*	2	220	*y*	2	220	*y*	2	220
9	*-x*	-	40	*-x*	-	40	*-x*	-	40
10	*x*	3	120	*x*	3	120	*x*	3	120
11	*x*	4	160	*x*	4	160	*x*	4	160
12	*x*	4	160	*x*	4	160	*x*	4	160
13	0	0	0	*y*	3	220	*-z*	-	140
14	0	0	0	*y*	3	70	*z*	-	140
15	0	0	0	*-y*	5	120	*-z*	3	120
16	0	0	0	*z*	-	30	*z*	3	120
17	0	0	0	*-z*	2	100	*y*	-	80
18	0	0	0	0	0	0	*-z*	3	150
19	0	0	0	0	0	0	*z*	3	150

Objective function

For optimisation purposes, five objectives are used to evaluate the assembly sequence. In this case, the mean of fitness value from different models is used. For mixed-model table vice assembly problems with G=3 models, the optimisation objectives as used in Chapter 7 are listed below.

f_1= Minimise number of direction change mean

f_2= Minimise number of tool change mean

f_3= Minimise mean of cycle time

f_4= Minimise number of workstations

f_5= Minimise mean of workload variation

The details of these objective functions are presented in Section 7.2.1.

The objective functions f_1 up to f_5 are calculated individually for every assembly sequence generated by the MODPSO algorithm. For example, consider an assembly sequence F_1[16-1-2-4-6-9-11-5-3-7-8-17-19-14-13-18-15-10-12]. The first step to evaluate this sequence is to assign assembly task into workstations. The task assignment for this sequence is shown in Table 8.32. For a given maximum cycle time, ct_{max}^{A} = 420, ct_{max}^{B} = 450 and ct_{max}^{C} = 470 seconds, the first workstation consists of tasks 16, 1, 2, 4, 6 and 9 that fulfilled the ct_{max} constraint for all models.

If task 11 is included in the first workstation, the ct_{max} constraint will be violated for all models. On the other hand, in workstation 3 which contains tasks 8, 17 and 19, if the next task (i.e. task 14) is assigned to this workstation, it still fulfils the ct_{max} constraint for model A and B. However the assignment of task 14 to workstation 3 will violate the ct_{max} constraint for model C. Therefore, the task 14 is assigned to the next workstation.

Table 8.32: Assembly task assignment example for mixed-model problem

*WS	1						2				3			4			5		
F_1	16	1	2	4	6	9	11	5	3	7	8	17	19	14	13	18	15	10	12
D_A	0	*z*	*-z*	*-y*	*-x*	*-x*	*x*	*x*	*-y*	*y*	*y*	0	0	0	0	0	0	*x*	*x*
T_A	0	-	2	-	-	-	4	-	-	-	2	0	0	0	0	0	0	3	4
M_A	0	30	160	40	50	40	160	80	50	30	220	0	0	0	0	0	0	120	160
D_B	*z*	*z*	*-z*	*-y*	*-x*	*-x*	*x*	*x*	*-y*	*y*	*y*	*-z*	0	*y*	*y*	0	*-y*	*x*	*x*
T_B	-	-	2	-	-	-	4	-	-	-	2	2	0	3	3	0	5	3	4
M_B	30	30	160	40	50	40	160	80	50	30	220	100	0	70	220	0	120	120	160
D_C	*z*	*z*	*-z*	*-y*	*-x*	*-x*	*x*	*x*	*-y*	*y*	*y*	*y*	*z*	*z*	*-z*	*-z*	*-z*	*x*	*x*
T_C	3	-	2	-	-	-	4	-	-	-	2	-	3	-	-	3	3	3	4
M_C	120	30	160	40	50	40	160	80	50	30	220	80	150	140	140	150	120	120	160

*WS: Workstation

Based on Table 8.32, the f_1 can be calculated by summing the direction changes for all models and then dividing by the number of models. For example in model A, the numbers of assembly direction changes in each workstation are 3, 2, 0, 0 and 0. The total number of direction changes for models A, B and C are 5, 7 and 8 respectively. Therefore, the first objective for this sequence is as follows:

$$f_1 = \frac{(5+7+8)}{3} = 6.67$$

The second objective is also calculated using a similar approach with f_1. For model A, the numbers of tool changes for each workstation are 2, 1, 0, 0 and 1, while the total direction changes for each model are 4, 5 and 8.

$$f_2 = \frac{(4+5+8)}{3} = 5.67$$

The third objective is calculated by calculating the cycle time for each model. Based on Table 8.32, the cycle time for models A, B and C are ct^A = 320, ct^B = 400 and ct^C = 450 seconds. Therefore the third objective for this sequence is calculated as follows.

$$f_3 = \frac{(320+400+450)}{3} = 390$$

Meanwhile the fourth objective can be determined directly once the assembly task assignment is completed. The total number of workstations for this sequence is 5.

$$f_4 = 5$$

Finally, the fifth objective can be calculated using Eq. 7.4, which is the mean of workload variation. The workload variation for model A (v_A), B (v_B) and C (v_C) are calculated as follows.

$$v_A = \frac{(320-320)+(320-320)+(320-220)+(320-0)+(320-280)}{5} = 92$$

$$v_B = \frac{(400-350)+(400-320)+(400-320)+(400-290)+(400-400)}{5} = 64$$

$$v_C = \frac{(450-440)+(450-320)+(450-450)+(450-430)+(450-400)}{5} = 42$$

Therefore the fifth objective can be calculated as follows.

$$f_5 = \frac{(92+64+42)}{3} = 66$$

Optimisation Results

The optimisation of this problem is run with 1000 iterations and 10 repetitions with different random seeds. The results acquired by all algorithms are then filtered to identify the Pareto optimal solutions. The summary of mixed-model results is presented using performance indicators as shown in Table 8.33.

Table 8.33: Comparison of performance indicators for mixed-model vice assembly

Algorithm	$\tilde{\eta}$[1]	ER[2]	GD[2]	$Spacing$[2]	$Spread_{max}$[1]	Weight sum	Rank
MOGA	13 (5)	0.4583 (6)	0.8504 (7)	8.4571 (3)	118.9272 (2)	23	2
ACO	0 (3)	1.0000 (3)	5.2545 (1)	4.3886 (7)	120.8318 (3)	17	5
HGA	16 (6)	0.8298 (4)	2.1113 (4)	5.3679 (5)	125.8010 (4)	23	2
NSGA-II	3 (4)	0.7857 (5)	1.7953 (5)	10.8595 (1)	116.7419 (1)	16	7
MOPSO	0 (3)	1.0000 (3)	2.2668 (3)	4.7499 (6)	125.8650 (5)	20	4
DPSO	0 (3)	1.0000 (3)	4.0663 (2)	8.6781 (2)	137.3685 (7)	17	5
MODPSO	25 (7)	0.4186 (7)	1.2150 (6)	6.8305 (4)	130.9052 (6)	30	1

[1] Larger the better indicator [2] Smaller the better indicator

*Number in brackets are weighting values from the best (weight=7) to worst (weight=1)

Based on Table 8.33, the proposed MODPSO came out with better performance in number of non-dominated solutions in Pareto optimal ($\tilde{\eta}$). Meanwhile in the GA-based algorithms, i.e. HGA, MOGA and NSGA-II, are in second, third and fourth places respectively. The MODPSO also performed better compared to other algorithms in the *ER* indicator. However, in the *GD* indicator, the MODPSO algorithm is defeated by the MOGA. Meanwhile, the

ACO algorithm performed better compared to other algorithms for the *Spacing* indicator. Finally, in the $Spread_{max}$ indicator, the best algorithm is DPSO, while MODPSO is the second-best. The summation of weight in the seventh column shows the overall algorithms' performance. From these numbers, the best overall algorithm to optimise the mixed-model vice assembly is the proposed MODPSO. In the second rank, two algorithms sharing similar weight summation are MOGA and HGA.

The non-dominated solutions for integrated mixed-model ASP and ALB of table vice that were found using MODPSO algorithm are presented in Table 8.34. In total, 43 non-dominated solutions are found from the optimisation. The results show that the minimum f_1 (mean of assembly direction change) is 5.00, minimum f_2 (mean of assembly tool change) is 1.67, minimum f_3 (mean of cycle time) is 336.67, minimum f_4 (number of workstation) is 5 and minimum f_5 (mean of workload variation) is 49.33.

Table 8.34: Non-dominated solution for mixed-model table vice assembly

No	f_1	f_2	f_3	f_4	f_5	Assembly sequence
1	5.00	4.67	413.33	6.00	143.33	16-1-4-9-6-2-5-12-11-3-7-8-17-18-19-14-13-15-10
2	5.00	5.67	413.33	5.00	89.33	16-1-9-5-11-3-4-7-8-17-2-13-18-6-14-19-15-12-10
3	5.33	4.33	413.33	6.00	143.33	16-1-4-9-6-2-11-12-5-3-7-8-17-18-19-14-10-15-13
4	5.33	4.67	413.33	5.00	89.33	16-1-4-9-6-2-11-12-5-3-7-8-17-14-19-10-15-18-13
5	5.33	5.00	403.33	5.00	79.33	16-1-9-5-11-3-4-6-7-8-12-10-17-13-14-19-15-18-2
6	5.67	3.67	380.00	6.00	110.00	16-1-9-5-11-3-4-7-8-2-17-13-14-6-12-19-18-15-10
7	5.67	4.00	416.67	5.00	92.67	16-1-4-9-6-2-11-12-5-3-7-8-17-14-13-10-15-18-19
8	5.67	4.33	413.33	5.00	89.33	16-1-4-6-9-2-11-12-5-3-7-8-17-14-13-18-19-10-15
9	5.67	5.33	360.00	6.00	90.00	16-1-9-5-11-3-4-6-2-12-10-7-8-17-13-19-14-15-18
10	6.00	3.33	420.00	6.00	150.00	16-5-4-6-9-12-11-3-7-8-17-13-19-14-1-2-15-18-10
11	6.00	4.67	373.33	6.00	103.33	5-4-9-6-1-16-12-11-3-10-7-8-17-14-19-13-2-15-18
12	6.33	2.33	410.00	6.00	140.00	16-1-4-9-6-5-11-12-3-7-8-17-13-14-19-10-18-15-2
13	6.33	2.67	403.33	6.00	133.33	16-1-4-9-6-5-11-12-3-7-8-17-2-13-10-19-14-15-18
14	6.33	3.33	370.00	6.00	100.00	16-1-4-9-6-2-11-5-3-7-8-17-13-14-19-10-15-18-12
15	6.33	5.00	353.33	6.00	83.33	16-5-4-6-9-11-3-7-8-17-13-1-19-14-15-2-18-12-10
16	6.33	5.33	400.00	5.00	76.00	16-1-4-6-9-11-3-5-7-8-10-12-17-14-19-13-2-18-15
17	6.33	7.00	340.00	7.00	108.57	9-11-16-3-4-10-5-7-8-6-12-17-18-14-19-1-2-15-13
18	6.67	2.67	363.33	6.00	93.33	16-1-4-5-6-9-11-12-2-3-7-8-17-18-19-14-13-10-15

No	f_1	f_2	f_3	f_4	f_5	Assembly sequence
19	6.67	4.00	343.33	6.00	73.33	16-5-4-9-6-11-1-3-7-12-10-8-17-13-14-19-15-2-18
20	6.67	5.67	390.00	5.00	66.00	16-1-2-4-6-9-11-5-3-7-8-17-19-14-13-18-15-10-12
21	6.67	7.33	373.33	5.00	49.33	16-1-2-4-6-9-5-11-3-7-8-17-19-10-12-14-15-18-13
22	7.00	2.33	356.67	6.00	86.67	16-4-1-5-9-6-12-11-2-3-7-8-17-13-14-19-18-15-10
23	7.00	3.00	420.00	5.00	96.00	5-4-3-9-1-16-12-11-3-7-8-17-19-10-14-13-2-15-18
24	7.00	4.00	413.33	5.00	89.33	4-5-9-6-1-2-11-12-16-3-7-8-17-13-19-14-15-18-10
25	7.00	4.33	403.33	5.00	79.33	16-1-9-5-11-3-4-6-7-8-17-14-12-13-18-10-15-19-2
26	7.00	4.67	386.67	5.00	62.67	16-1-4-9-6-2-11-5-3-7-8-17-14-19-18-13-15-10-12
27	7.00	6.33	383.33	5.00	59.33	16-1-2-4-6-9-11-5-3-7-8-17-14-15-18-13-19-12-10
28	7.00	7.00	373.33	5.00	49.33	4-1-16-6-9-2-5-11-3-7-8-17-19-14-12-10-18-15-13
29	7.33	3.00	413.33	5.00	89.33	16-1-4-9-6-5-11-12-3-7-8-17-14-10-18-13-2-15-19
30	7.33	4.00	383.33	5.00	59.33	16-1-4-6-9-12-11-5-3-7-8-17-14-13-10-19-2-18-15
31	7.33	6.00	373.33	5.00	49.33	16-1-4-9-6-11-3-5-12-7-8-17-19-18-13-10-14-15-2
32	7.67	2.00	383.33	6.00	113.33	16-1-9-4-6-5-12-11-2-3-7-8-17-13-14-10-15-18-19
33	7.67	3.67	350.00	6.00	80.00	16-4-1-5-6-9-11-3-7-8-17-13-12-10-14-19-18-15-2
34	7.67	5.33	376.67	5.00	52.67	9-1-16-5-11-3-4-6-7-12-8-17-18-10-13-19-14-15-2
35	7.67	5.67	373.33	5.00	49.33	1-16-2-4-6-9-11-5-3-7-8-17-19-13-18-10-14-15-12
36	8.00	4.67	373.33	5.00	49.33	16-1-4-6-9-11-3-7-5-12-8-17-19-18-10-13-2-14-15
37	8.00	6.33	340.00	6.00	70.00	16-1-9-4-5-6-11-12-3-7-2-8-17-14-13-19-18-15-10
38	8.33	1.67	406.67	6.00	136.67	16-1-8-4-5-6-11-12-3-7-2-8-17-14-13-19-18-15-10
39	8.33	3.67	343.33	6.00	73.33	16-4-1-5-6-9-11-3-7-8-17-12-14-18-13-19-10-2-15
40	8.33	5.33	340.00	6.00	70.00	4-1-9-11-2-3-16-6-10-5-7-8-17-13-14-19-18-12-15
41	8.67	3.67	410.00	5.00	86.00	5-4-6-9-1-2-12-11-3-7-8-16-17-10-14-13-19-18-15
42	9.33	5.33	336.67	6.00	66.67	16-5-4-6-9-11-3-7-8-17-13-19-10-18-1-2-12-14-15
43	9.67	3.67	336.67	6.00	66.67	16-5-4-6-1-9-11-3-7-8-17-13-19-10-2-14-15-12-18

The non-dominated solutions in Table 8.34 are plotted in matrix plot, as shown in Figure 8.21. The non-dominated solution spread from MODPSO can be observed in diagonal boxes. For the f_1, the non-dominated solution is spread uniformly with 0.33 gap between solutions except between solutions 41 and 42, which have a larger gap (i.e. 0.66). On the other hand, the f_2 is also spread uniformly with similar 0.33 gaps, but with slightly larger gaps between solutions 28 and 37.

Meanwhile for f_3, there are two spotted points that have different gaps compared to other solutions. The first point is between solutions 14 and 18,

while the second point is between solutions 16 and 20. The gaps for both points are 6.67 and 10 respectively, while the gaps for all other adjacent solutions are 3.33. The f_4 objective is evenly spread from 5 to 7 workstations. For this objective, the theoretical minimum number of workstations (i.e. 5) is fully achieved. This number is acquired from the largest theoretical number of workstations among different models. It is calculated by dividing the total assembly task time for a particular model by the maximum cycle time for that model. In the meantime, the adjacent gaps of f_5 are intermixed mostly between 2.67 and 6.67. The largest gap for this objective is 20, found between solutions 13 and 32.

Based on Figure 8.21, the non-dominated solution for the integrated mixed-model table vice assembly is mostly spread uniformly with the exception as stated above. This result shows that although it still has room for further improvement, the MODPSO algorithm is able to explore the solution space.

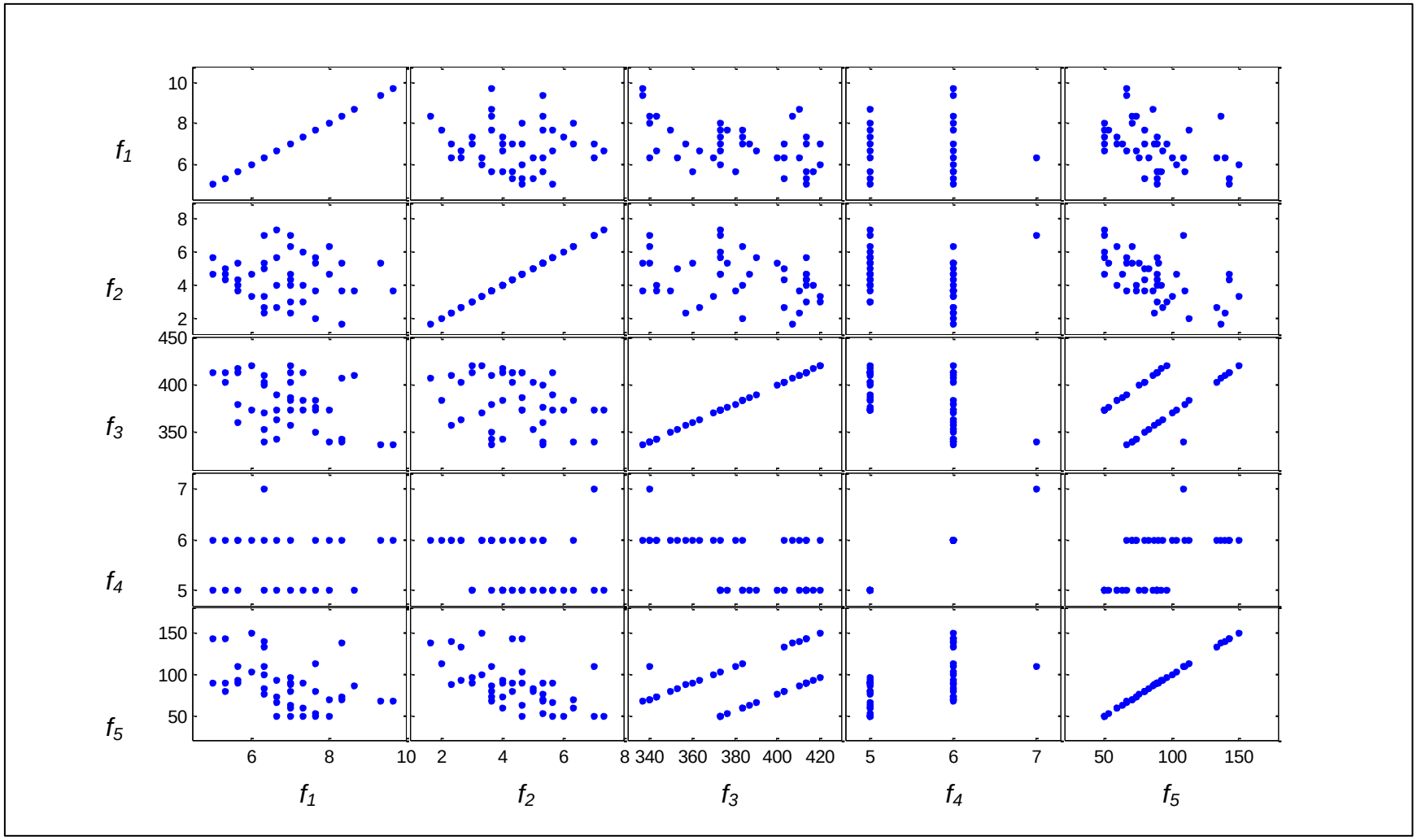


Figure 8.21: Non-dominated solution spread for mixed-model table vice using MODPSO

8.3.5 Summary of Validation using Real-World Problems

This section presents four real-world assembly problems. The problem representation as presented in Chapter 4 is explained, starting from assembly liaison identification, precedence diagram and assembly data establishment. These problems are then optimised using the proposed MODPSO algorithm. For comparison purposes, six other algorithms are also used to optimise these problems.

The optimisation result of the fixed table vice assembly which represents low difficulty problems shows the benefit of integrated ASP and ALB optimisation. In this case study, if the optimisation of ASP and ALB is performed sequentially, only one non-dominated solution from ASP will be found, resulting in no other option for the successor activity (i.e. ALB). However, when the ASP and ALB optimisation are integrated, the non-dominated solutions show that there are a few other options that come out with better ALB results by compromising the number of direction or tool changes.

The second real-world problem that involved assembly of toy train is the assembly problem with medium number of parts and tasks. The highlighted result from this problem is the ability of MODPSO algorithm to balance the workload with less than 1 second in every workstation with minimum number of workstation.

Meanwhile, the third real-world problem from Bautista and Pereira (2007), with 140 tasks, represents assembly problems with high number of tasks. The algorithm performance shows that the proposed MODPSO and NSGA-II show better overall performance compared to the ACO used in the original article. The MODPSO algorithm also presents better line balancing results compared to the result published in the original article.

Finally, in the real life mixed-model assembly problem, the different table vice models are represented using a joint precedence diagram. The nearly uniform

non-dominated solution spreads from the optimisation, indicating the ability of the MODPSO algorithm to explore the search space for real life problem.

The overall algorithm performance which was determined based on summation of the assigned weight consistently shows that the proposed MODPSO algorithm is in the first rank for all problems. From all problems presented in this section, it can be summarised that the proposed MODPSO algorithm can be used to optimise real assembly problems, using the integrated task-based representation. Besides that, it also shows the benefit of integrated ASP and ALB optimisation.

8.4 Comparison of Sequential and Integrated ASP and ALB Optimisation

This section will validate the advantages of the integrated optimisation approach by comparing it to the sequential approach. The details of sequential and integrated optimisation approaches for ASP and ALB is presented in Section 6.2. For comparison purposes, the solution quality towards Pareto optimal from both optimisation approaches will be used. Therefore, five objective functions, as used in Section 6.3.1, will be adopted. The objective functions are to:

f_1 = minimise number of assembly direction change (n_{dc})

f_2 = minimise number of assembly tool change (n_{tc})

f_3 = minimise cycle time (ct)

f_4 = minimise number of workstations (nws)

f_5 = minimise workload variation (v)

For optimisation purposes, the MODPSO algorithm will be used in both optimisation approaches. In order to test both optimisation approaches, the test problems generated from the tuneable test problem generator are used. In this test, a similar experimental design as used in Table 6.6 is applied. It consists of

51 test problems with a different range of tuneable input (i.e. number of tasks (*n*), Order Strength (*OS*), Time Variability Ratio (*TV*) and Frequency Ratio (*FR*)). The optimisation of each of test problem is run with 500 iterations.

8.4.1 Sequential and Integrated Optimisation Results

Figure 8.22 presents the plot of performance indicators for sequential and integrated optimisation approaches. For the number of non-dominated solutions in Pareto optimal ($\tilde{\eta}$) and maximum spread indicators, a larger value shows better performance, while for remaining indicators, a smaller value presents better performance. Based on Figure 8.22, the $\tilde{\eta}$ indicator consistently indicates that the integrated optimisation approach presents larger numbers of non-dominated solutions in Pareto optimal compared to the sequential approach.

Meanwhile the results of *ER*, *GD* and *Spacing* indicators show the intermixed performance between sequential and integrated approaches. For *ER* and *GD* indicators, the sequential optimisation approach performed better than the integrated approach in 69% and 67% of the problems respectively. From these percentages, around 86% are problems with medium and large sizes. On the other hand for the *Spacing* indicator, the sequential optimisation approach performed better in 41% of the test problems. However, no specific problem category is identified for the problem when sequential optimisation performed better. Finally, for the $Spread_{max}$ indicator, the integrated optimisation shows better performance in 96% of the test problems.

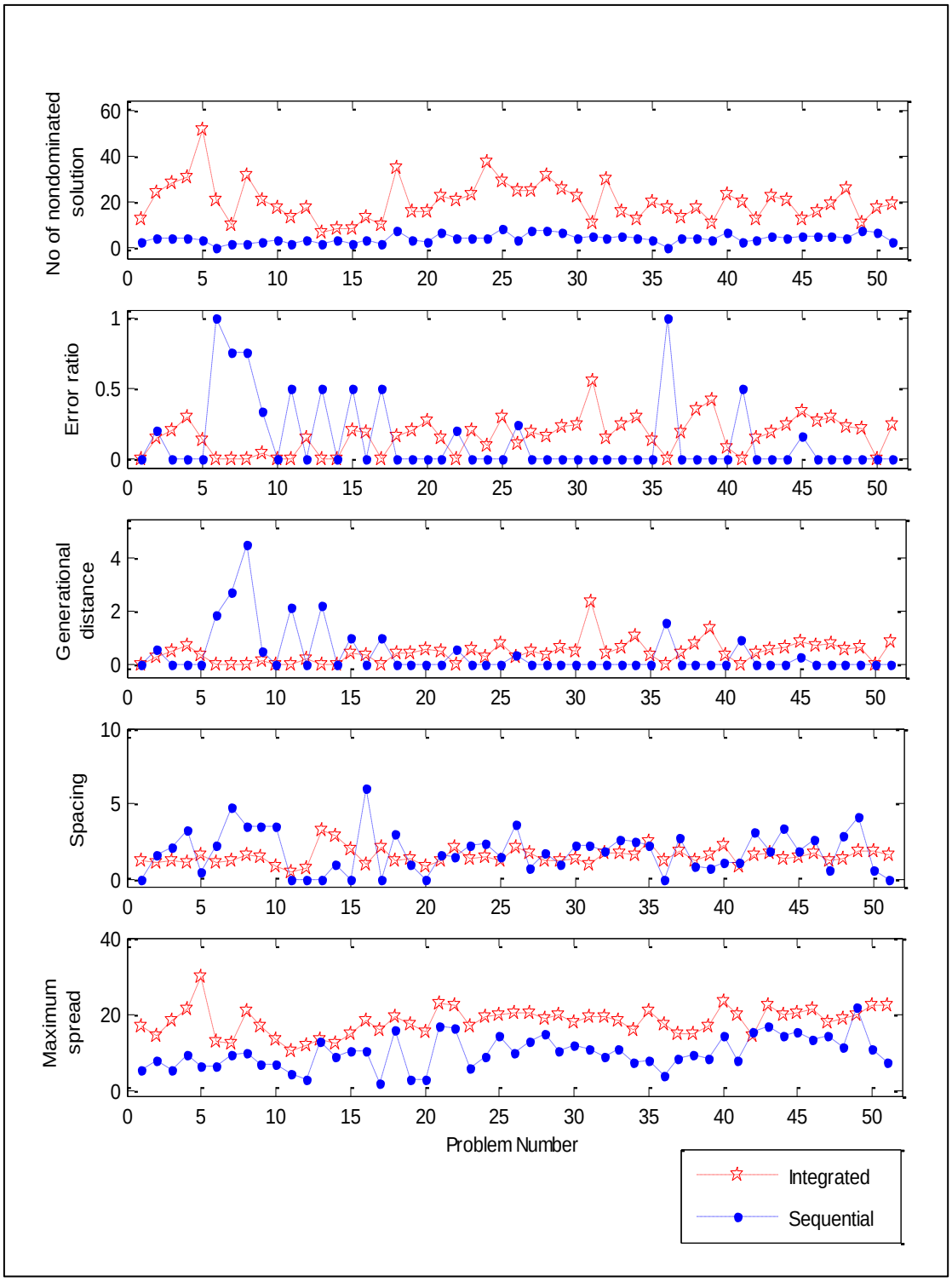


Figure 8.22: Plot of performance indicators for sequential and integrated optimisation

Besides the performance indicators above, another important measure when comparing the performance of sequential and integrated optimisation approaches is the minimum objective function values that were achieved using both approaches. Figure 8.23 shows the plot of minimum objective function values acquired by sequential and integrated optimisation.

Based on Figure 8.23, the integrated optimisation approach has achieved better or equal minimum direction change objective value in 67% of the test problems. Meanwhile, for the assembly tool change objective, the results show that the integrated optimisation approach comes out with better or equal minimum value in 61% of the test problems. The majority of these problems are problems of small and medium size. On the other hand, for ALB objective functions, the integrated optimisation approach shows better or equally minimum objective values in all test problems compared to the sequential approach. This finding is consistent for the objective of minimising cycle time, number of workstations and workload variation.

From the ASP objectives, the integrated optimisation performs better in small and medium size problems, while the sequential optimisation has better minimum objectives in large size problems. This is related to the excessively large search space for large size problems. In order to prove this hypothesis, a further test for large size problems is conducted. It is performed by running the sequential and integrated optimisation with larger iteration numbers (until 3000 iterations). For this purpose, the test problems with 80 tasks are selected. The optimisation result of this test is presented in Figure 8.24. In general, the integrated optimisation approach is able to come out with minimum ASP objectives value as is the sequential approach, by running the optimisation between 4 to 6 times longer than earlier runs.

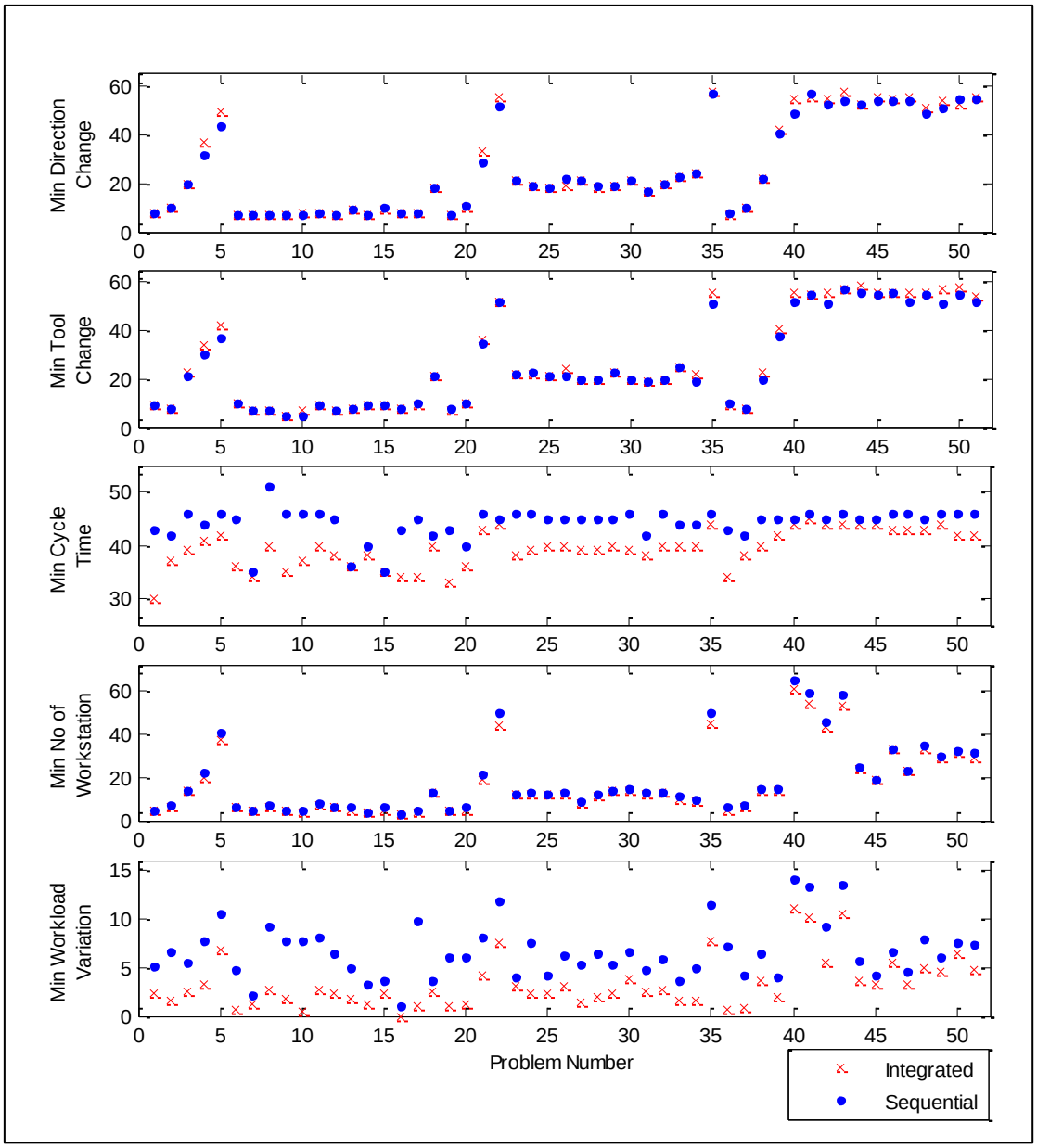


Figure 8.23: Plot of minimum objective function values

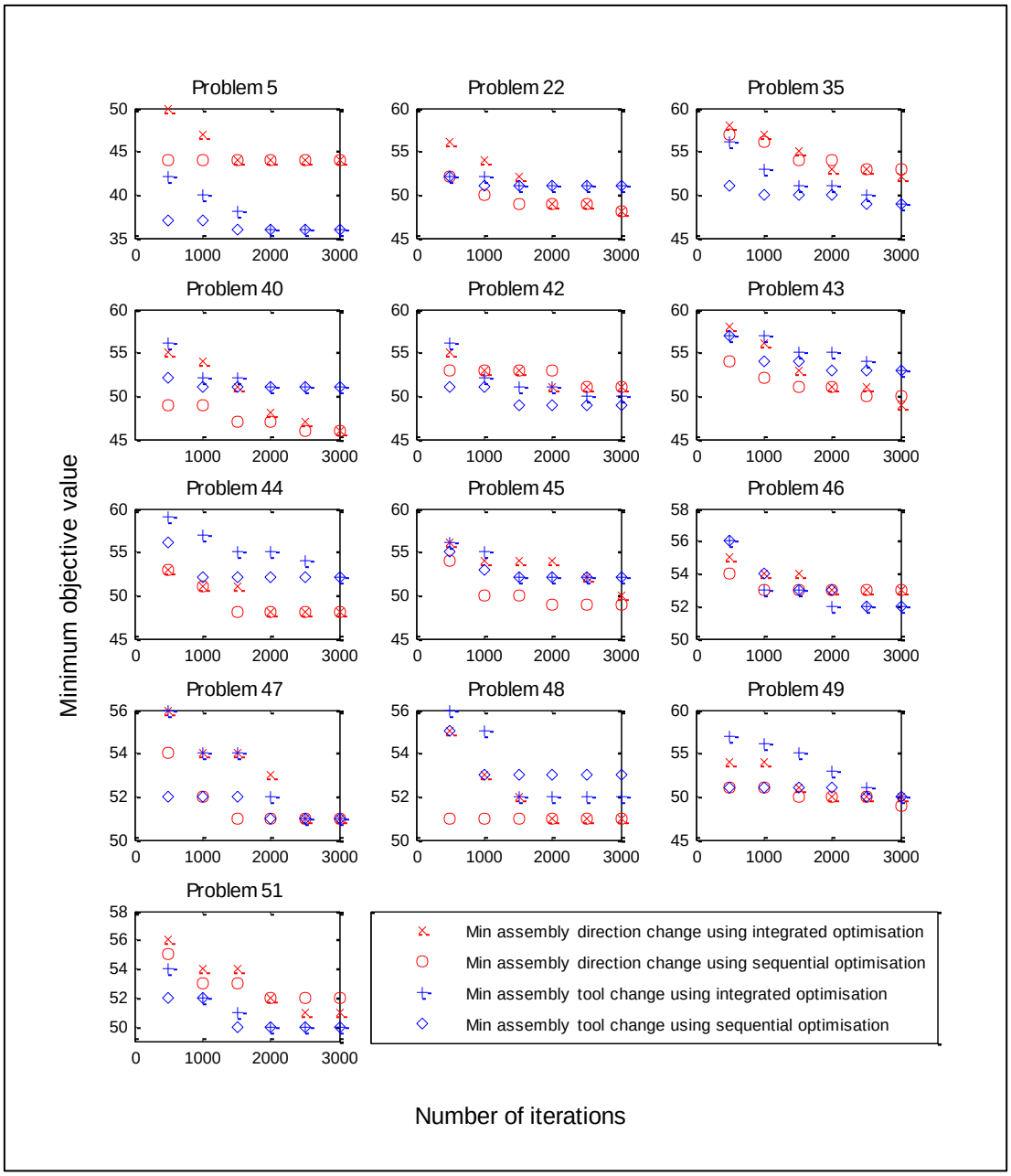


Figure 8.24: Plot of minimum ASP objectives with larger iteration

8.4.2 Discussion of Sequential and Integrated Optimisation Results

The result of *ñ* shows that the integrated optimisation approach is able to come out with larger number of solutions in the Pareto optimal compares with the sequential approach. This result is related to the size of ASP and ALB search space for the integrated optimisation approach. In this approach, the ASP and ALB share similar size of search space. Compared to the sequential optimisation approach, the search space for ALB is vastly reduced compared to ASP, since it is formed by the non-dominated solutions of ASP. In this case, the search space for integrated ASP and sequential ASP is similar, but the search space for sequential ALB is much smaller compared to integrated ALB. Therefore, the chance to produce better solutions for the integrated optimisation approach is much higher compared to the sequential approach.

On the other hand, the mixed performance of the integrated approach in *ER* and *GD* is because of the number of non-dominated solutions found using this approach. Although the number of non-dominated solutions found using the sequential approach is much smaller than for the integrated approach, the chances of the solutions to be in Pareto optimal are better. This is because in the first part of the sequential approach (i.e. ASP optimisation), only two objective functions are evaluated, while the remaining objective functions are evaluated in the second part (i.e. ALB optimisation). Therefore, the chances to converge to ASP non-dominated solutions that were in Pareto optimal solutions are better.

This reason is supported by the results of the minimum objective values as presented in Figure 8.23. In some of the problems, the ASP objective values using the sequential approach is better than the integrated approach, but the ALB objectives values using integrated optimisation are consistently better or equal than with the sequential approach. This reason is also applied to the *GD* indicator. The *GD* is the measure of average distance of non-dominated solutions to the nearest Pareto optimal solution. Although the number of non-dominated solutions using the sequential approach is less than for the

integrated approach, most of the found solutions are in Pareto optimal because of there being fewer objective functions in the first part of optimisation. For this reason, the *GD* of sequential optimisation approach is found to be better than the integrated approach in more than 60% of the problems.

Meanwhile, the *Spacing* indicator that represents the solution uniformity between one solution and another nearest solution, shows mixed performance of sequential and integrated optimisation approaches for the test problems. However, the result did not show any specific problem category in which integrated or sequential approach performed better. This indicator depends on the number of solutions found and also the solution spread. For similar solution spread, the optimisation approach which comes out with the larger number of non-dominated solutions will have better *Spacing*.

In addition, the $Spread_{max}$ indicator which shows the capability of exploring extreme solutions indicates that the integrated optimisation approach shows better results compared to the sequential approach. The integrated optimisation shows better $Spread_{max}$ in 49 out of 51 test problems (96%). This finding is closely related to equal search space for ASP and ALB in integrated optimisation, where the chances of finding better extreme solutions especially in ALB is much better compared to the sequential approach.

The experiment also measured the minimum objective values which were achieved by sequential and integrated approaches for each test problem. The results show that in the ASP objectives (to minimise assembly direction change and tool change), the integrated optimisation approach has equal or better minimum values in small and medium size of problems. Meanwhile, the sequential approach has better minimum value in large size problems. This finding is due to the complexity of large size problems and also the way integrated optimisation works. In integrated optimisation, all five objective functions are evaluated together, which influences the particle convergence towards the best solution for both ASP and ALB. Compared to the sequential optimisation which only evaluates ASP objectives in the first part, the particles

will directly converge to the ASP as best solution, which results in a better chance of coming out with better ASP objectives value.

Although the sequential optimisation approach has greater chances to generate better minimum objective values for ASP, it only happened with large sizes of problems. This is because of the complexity of large size problems, where the search space for larger problem size is excessively increased when the number of tasks increases (Qin et al., 2007; Lv et al., 2010). In small and medium sized problems, both optimisation approaches were able to explore the entire search space, so equally minimum objective values resulted in these categories. However, in large size problems, the integrated approach has limited performance due to the extremely large search space. Therefore, the optimisation approach that focuses on fewer objectives will have a larger chance of generating better minimum objective values.

In order to prove this hypothesis, a further test for large size problems is conducted. It is performed by running the sequential and integrated optimisation for longer periods of time. The result from this test indicated that for large size problems, the integrated optimisation approach is able to come out with minimum ASP objectives value as in the sequential approach by running the optimisation between 4 to 6 times longer than in the earlier runs. It shows that the integrated optimisation is able to explore the search space but needed more iteration for large size problems.

On the other hand, in ALB objectives, the integrated optimisation approach consistently shows better or equal minimum values compared to sequential optimisation. The integrated optimisation presents better minimum values in 88% of the problems, while in the remaining 12% the integrated and sequential optimisation approaches share similar minimum objective values. This result is also because of larger search spaces of integrated ALB compared to sequential ALB. Therefore, the chances of finding better minimum solutions are greater in the integrated optimisation approach.

8.5 Chapter Summary

The objective of this chapter is to validate the proposed MODPSO algorithm. Before proceeding with the validation work, the problem selection that covers low, medium and high problem sizes were explained.

The first part of validation is performed by testing the MODPSO algorithm on the selected artificial test problems published in the literature. The optimisation results from the MODPSO algorithm is then compared to the results from MOGA, ACO, HGA, NSGA-II, MOPSO and DPSO algorithms. The MODPSO's result is also compared to the published results from the original sources. The experimental result concluded that the proposed MODPSO was able to search for better solutions compared to the published results in the original sources either in terms of Pareto solution numbers or better line balancing.

The second section of this chapter validates the ability of the proposed MODPSO to optimise real-world problems. Four real-world problems have been presented and optimised using the MODPSO algorithm. The optimisation results concluded that the proposed MODPSO algorithm shows better overall performance compared with comparison algorithms. The results from this section also concluded that the non-dominated solution from MODPSO is mostly spread evenly for the number of direction change, number of tool change, cycle time and number of workstation objectives. Meanwhile for workload variation objective, the non-dominated solution spread is varied due to workload variation equation.

The third section of this chapter compares the sequential and integrated optimisation in terms of solution quality towards the Pareto optimal. The optimisation results indicate that the integrated optimisation approach has advantages in finding more non-dominated solutions and exploring extreme solutions in all problems. On the other hand, the sequential optimisation approach has better accuracy of solutions in Pareto optimal although the number of non-dominated solutions in Pareto optimal is less than the integrated optimisation approach, due to part by part optimisation in the sequential

approach. Based on the findings from this section, the integrated optimisation approach is appropriate to be used for small to medium ASP size problems and in all ALB problems; while for the large size ASP problems, the integrated optimisation approach required more iterations (between 4 to 6 times) to reach similar minimum solutions as in the sequential ASP approach.

Finally, the MODPSO algorithm has consistently shown better overall performance for all seven problems presented in Sections 8.2 and 8.3. In conclusion, this chapter had achieved the following goals:

- The problem selection used for validation has been explained.
- The ability of the MODPSO algorithm to search for better solutions compared to published solutions from the literature has been indicated.
- The ability of the MODPSO algorithm to perform better than comparable algorithms in real-world problems has been indicated.
- The advantages of the integrated optimisation approach and sequential optimisation approach has been compared.

CHAPTER 9

DISCUSSION AND CONCLUSIONS

This chapter discusses the key findings and observations of this research. It discusses the findings as a platform to identify the contributions in this research. Later, the research aim and objectives achievement is discussed.

9.1 Key Observations

This section summarises the key observations from this research. The discussion is based on the main stages of the research.

9.1.1 Literature Review

The literature review on assembly optimisation reveals that the optimisation activities are classified according to product development and production stages as follows: (i) Product Conception and Design stage, (ii) Production Planning stage and (iii) Manufacturing Process stage. From the literature, numerous examples of research on integrated assembly optimisation activities between stages (i) and (ii) have been published, but not much work has been conducted to optimise integrated optimisation activities in (ii) and (iii), i.e. Assembly Sequence Planning (ASP) and Assembly Line Balancing (ALB).

The literature survey in this research focuses on soft computing approaches to optimise ASP and ALB. Based on the survey, two most frequently used ASP

optimisation objectives are to minimise (i) number of direction changes and (ii) number of assembly tool changes. Meanwhile in ALB, the main optimisation objectives are to minimise (i) cycle time, (ii) number of workstations and (iii) workload variation.

On the other hand, various soft computing techniques have been used to optimise ASP and ALB individually ranging from Evolutionary Computation to Neural Network approach. From the survey, three algorithms which accumulated more than 70% of the publications on individual ASP and ALB optimisation are Genetic Algorithms (GA), Ant Colony Optimisation (ACO) and Particle Swarm Optimisation (PSO).

In the meantime, the PSO algorithm which is relatively new compared to GA shows greater acceptance by the researchers with incremental popularity over the studied years. In a number of published ASP and ALB works in which the GA and PSO algorithms' performance are compared, PSO was proved to have better overall performance compared to GA (Section 2.4.5). Although there are many further discussions on this matter, it is worth investigating this claim on integrated ASP and ALB application because of the growth in number of publications using PSO in their research which indirectly explains the ability of this algorithm in ASP and ALB optimisation.

The optimisation algorithm of integrated ASP and ALB in the literature is currently limited to GA, which is generally known as an efficient and robust algorithm for optimisation. The PSO algorithm, which indicates better acceptance over the years, presents a good opportunity to be implemented based on its track record.

In the literature, researchers have successfully developed various ASP representation schemes to fit their particular problem characteristics and attributes. One of the common similarities between these schemes is that they are built based on assembly parts. On the other hand, the most dominant and successful representation method in ALB is the precedence graph, which is built based on the assembly task. In order to optimise integrated ASP and ALB, a

common representation built using a similar basis is required. In this case, the assembly task basis is chosen after considering the available alternatives, flexibilities and success of previous works. Although there is a work that uses a similar approach, this work does not clearly define how the assembly direction change is determined from a real assembly product (Tseng et al., 2008)

In the literature, assembly line problems are generally classified into simple and generalised assembly line problems (Scholl and Becker, 2006). The simple assembly line only runs one homogenous product on a serial line layout, and all workstations are equally equipped with machines and workers (Scholl and Becker, 2006). Meanwhile, the generalised assembly line includes all problems that are not simple assembly problems. One of the popular problems under generalised assembly is mixed-model assembly line problem (Tasan and Tunali, 2008). The existing optimisation of integrated ASP and ALB is currently limited to simple problems only.

The literature review has presented different research opportunities and gaps in integrated ASP and ALB optimisation. From the review and analysis of the current works, it is found that there is a dearth of test problems for integrated ASP and ALB problems with a wide range of difficulties. The current test problem generator is limited to generating only ALB test problems. Besides that, the integrated ASP and ALB optimisation works from the literature are limited to Genetic Algorithms, while PSO presents better performance in individual ASP and ALB optimisation. Finally, the current integrated ASP and ALB optimisation is limited to single-model problems. Therefore, an effort to formulate and optimise integrated ASP and ALB to more general types of problem definitely brings a significant contribution to knowledge.

9.1.2 Integrated ASP and ALB Representation

This research proposed an integrated ASP and ALB representation scheme to model an assembly product. In this research, the representation scheme needs

to represent the assembly precedence constraint and assembly data. Based on the main optimisation objectives from the literature survey, the required assembly data are assembly direction, assembly tool and assembly time.

The representation scheme is developed based on assembly tasks, which were commonly used in the ALB work. Therefore, it can be directly applied to represent an ALB problem. However, to implement the task-based representation to ASP, a new definition is needed for assembly direction since no existing work defines assembly direction on the assembly task basis. In this research, the assembly task is defined as assembly work that consists of two entities; 'part and part', 'part and subassembly' or 'subassembly and subassembly'. To define assembly direction in task-based representation, one part is defined as the moving part, while another part is the fixed part. Therefore, the assembly direction is identified from the direction of bringing the moving part to be assembled to the fixed part.

In contrast to existing task-based representations for integrated ASP and ALB, the proposed representation scheme considers the assembly direction as one of the pieces of information needed with regard to assembly. This is important because the assembly direction change is the most frequently used ASP objective from the literature survey, which shows the importance of this objective to be considered in optimisation work. Besides that, the proposed representation scheme also clearly defines how the assembly direction is determined from real-world assembly problem.

On the other hand, in comparison with connector-based representation, the task-based representation is closely linked to the actual assembly process. This is because in the task-based representation models the assembly process is based on assembly relations between two parts, while the connector-based representation modelled the assembly process by grouping the parts according to a connector. In this case, the number of parts grouped under one connector is unlimited. Besides that, the assembly direction information using task-based representation is more accurate compared to the connector-based because it

considers the direction of assembly parts (including connectors) rather than the direction of assembly connector only.

9.1.3 Development of Tuneable Test Problem Generator

In order to overcome the lack of test problems for integrated ASP and ALB in a range of difficulties, a tuneable test problem generator is developed. The test problem generator generates integrated ASP and ALB test problems with tuneable difficulty levels controlled by four tuneable inputs; number of tasks (*n*), Order Strength (*OS*), Frequency Ratio (*FR*) and Time Variability ratio (*TV*).

From the experiments, the problem difficulties will increase when using larger *n*, smaller *OS*, larger *FR* or smaller *TV*. The highlighted observations from the experiment in this stage contradict the *OS* effect on the problem difficulties in the literature. In some of the published papers, the larger *OS* leads to higher problem difficulty. This mismatch is due to the dissimilar approaches used in solving the precedence graph. In works that directly use generated permutation as an assembly sequence, precedence graphs with higher *OS* values are harder to solve. However in the works that ensure the feasibility of sequence such as using topological sort, the precedence graphs with higher *OS* value are easier to solve, because of differences in search space size.

Besides that, the statistical test is conducted to identify significant difficulty difference between one level and another. It shows that the selection of appropriate gaps for each tuneable input plays an important role in distinguishing the problem difficulties between different levels. Further experiments confirm that the problems generated by the TPG offer a sufficient range of problem variety to be used in algorithm testing. The generated problems are found to be useful in identifying the strengths and weaknesses of the tested algorithms.

9.1.4 MODPSO Algorithm for Integrated ASP and ALB

This research proposes a Multi-Objective Discrete Particle Swarm Optimisation (MODPSO) algorithm which is developed to optimise integrated ASP and ALB problems. The previous version of PSO algorithm applied discrete procedure to update position and velocity, but used a weighted approach to solve multi-objective problems. Another version of PSO uses the Pareto-based approach to solve multi-objective problems, but uses a standard mathematical operator to update position and velocity. In contrast to the existing algorithms, the proposed MODPSO which used the Pareto-based approach to deal with multi-objective problems adopted a discrete procedure instead of standard mathematical operators to update its position and velocity.

From experiments using test problems generated from the test problem generator, the MODPSO algorithm is shown to be capable of converging to Pareto optimal better than comparable algorithms. Statistical tests confirm that the MODPSO algorithm has presented significant improvements in terms of finding larger numbers of solutions and also better solution quality towards Pareto optimal. However, in terms of solution uniformity, the significant improvement achieved by MODPSO is only applied to certain comparable algorithms.

9.1.5 Integrated Mixed-Model ASP and ALB Optimisation using MODPSO

This research formulates and studies the optimisation of integrated mixed-model ASP and ALB problem. The mixed-model problem is formulated using a joint precedence graph which represents all the models in one precedence graph. In this approach, the precedence graphs representing different models are transformed into a joint precedence graph. Therefore, the MODPSO algorithm can be applied to optimise integrated mixed-model ASP and ALB with small modifications to the objective function.

For integrated mixed-model ASP and ALB, the experimental results indicate that the MODPSO algorithm performed better than other comparison algorithms. The MODPSO has shown significant improvements in converging to Pareto optimal solutions and exploring the extreme solutions in search space. However, the MODPSO algorithm does not perform well in terms of solution uniformity, as found in the single-model problem.

The analysis of results indicated that the MODPSO algorithm performs best in the problems with a high level of difficulty. Meanwhile, the weakest performance comes in the problem at low difficulty level, although it still performs better than comparable algorithms.

9.1.6 Comparison of Sequential and Integrated Optimisation Approaches

Validation of the integrated optimisation approach is conducted to confirm the capability of this approach. This is performed by comparing the integrated optimisation approach with normal sequential optimisation in terms of solution quality towards Pareto optimal. In the sequential approach, the ASP is optimised first, whilst the ALB is only started after ASP optimisation has been completed. In this approach, the non-dominated solutions of ASP are the search space for ALB.

The optimisation results of both approaches show that the integrated optimisation has advantages in finding more non-dominated solutions and exploring the extreme solutions. The optimisation results also indicate that for ASP, the integrated optimisation approach is able to come out with better or equal minimum objective values in small and medium sized problems compared to the sequential approach. On the other hand, the sequential optimisation approach shows better minimum ASP objective values in large size problems. This is due to the extremely large search space for large sized problems compared to small and medium sized problems, since the further test indicates that the integrated approach is able to catch up with a similar minimum as in the sequential approach, but with larger iterations (4 to 6 times larger).

However, the integrated optimisation approach consistently shows better minimum objective values for ALB in all test problems compared to the sequential approach. The optimisation results of sequential and integrated optimisation approaches clearly validate the advantages and disadvantages of the integrated optimisation approach in terms of solution quality towards Pareto optimal. It is also the general guideline in future to select appropriate optimisation approaches for a particular group of problems.

9.1.7 Validation using Artificial and Real-World Problems

The proposed MODPSO algorithm is validated using artificial test problems from the literature and also real-world problems. For validation using artificial test problems from the literature, the MODPSO's results were compared with the published results from the original sources. Besides that, the MODPSO results were compared with results from six comparable algorithms. The experimental results demonstrate that the proposed MODPSO is able to search for better solutions compared to the published results in the original sources either in terms of Pareto solution numbers or better line balancing. This finding was consistent for all test problems representing different problem sizes.

Meanwhile, for validation using real-world problems, four real-world problems have been presented and optimised using the MODPSO algorithm. The optimisation results conclude that the proposed MODPSO algorithm shows better overall performance compared to comparison algorithms. The results from this section also show that the non-dominated solution from MODPSO is mostly spread evenly for the number of direction changes, number of tool changes, cycle time and number of workstation objectives. Meanwhile, for the workload variation objective, the non-dominated solution spread is varied due to the workload variation equation. The results of validation confirm the capability of MODPSO to optimise multi-objective integrated ASP and ALB problems better than the published results and better than comparable algorithms.

9.1.8 Trends in Assembly Sequences

Further analysis of the results from artificial and real-world problems has led to the identification of trends in the assembly sequences of many of the considered problems. For this purpose, the tasks in precedence graph are classed based on the stage (or column) where a particular task belongs to. For a particular assembly task in precedence graph, the corresponding stage is equivalent to p_{max}+1. p_{max} refers to maximum number of precedence in serial for the task. For fixed table vice problem (Figure 8.6), the classes of assembly tasks, based on stages, are presented in Figure 9.1.

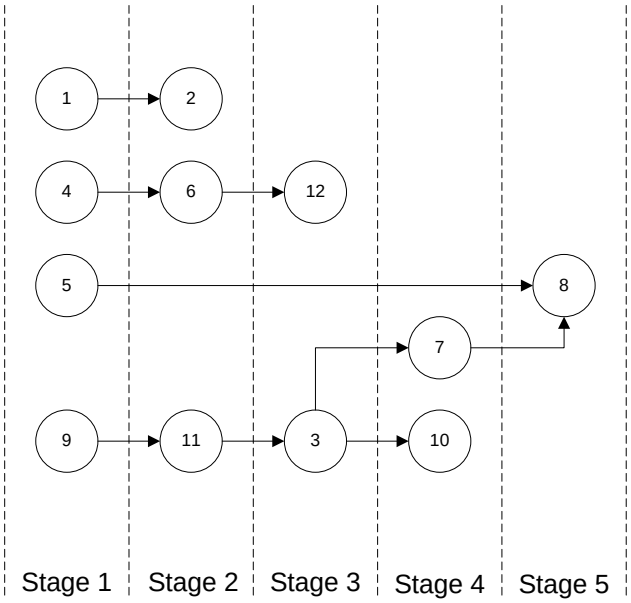


Figure 9.1: Precedence graph based on stage

One of the most popular heuristic approaches in ALB is Kilbridge and Wester approach (Hamza and Manna, 2013; Becker and Scholl, 2006). This approach solves a precedence graph stage by stage. The details of this approach can be found in Ponnambalam et al. (1999) and Hamza and Al-Manaa (2013). By using this approach, the assembly sequence for the fixed table vice problem is [1-4-5-9-2-6-11-3-12-7-10-8]. The objective function values are f_1=9, f_2=3, f_3 = 400, f_4 = 3 and f_5=20. This solution is not Pareto optimal because it is dominated by solution 21 in Table 8.18.

In contrast to Kilbridge and Wester's approach, the assembly sequence in Table 8.18 suggests a need to select assembly tasks from different stages, while maintaining the precedence feasibility. This can be observed from solution numbers 2, 7, 8, 9, 16, 17, 20 and 21 where the task selection across the stages occurs in early part of assembly sequences. Similar trends in assembly sequences are also seen in other problems that have been used for validation. For the artificial problem with 20 tasks (Figure 8.1), 40 tasks (Figure 8.2) and mixed-model problem (Figure 8.3), this trend is observed in 36% to 68% of the Pareto optimal solutions.

This trend is associated with the capability of the proposed algorithm to explore a large number of different feasible assembly sequences in the search space. One of the factors that influences this feature is the sensitivity of the encoding used to map unique feasible assembly sequences in solution space. In standard PSO (including MOPSO) and ACO algorithms, continuous real number encoding is used. In order to deal with the discrete feature in ASP and ALB, the encoded value is used as a weighting factor to determine the assembly task selection order from precedence graph. As a result, small changes in the encoded value do not affect the assembly sequences.

For example, let us consider the encoding in Table 9.1 for precedence graph in Figure 9.1. In this example, x_{i_t} represents particle *i* at iteration *t*. Based on the current approach, the x_{i_t} is the weighting factor for assembly task selection. To select the first task, the candidate tasks are 1, 4, 5 and 9 because these tasks do not have any precedence. With x_{1_1} as the weighting factor, task 9 is selected since it has the largest weight among the candidate tasks. By using this rule, the generated assembly sequence is [9-4-1-2-5-11-3-10-7-8-6-12]. In this case, when only small changes happen between x_{1_1} and x_{1_2}, the order of assembly sequence is not affected, since similar assembly sequence as above is generated.

Table 9.1: Example of real number encoding

Task	1	2	3	4	5	6	7	8	9	10	11	12
x_{1_1}	8.15	9.06	1.27	9.13	6.32	0.98	2.79	5.47	9.58	9.65	1.58	9.71
x_{1_2}	8.83	8.96	1.52	9.13	6.50	1.12	2.59	5.47	9.25	9.86	1.88	9.36

In comparison, the discrete encoding used in proposed MODPSO ensures that even small changes in algorithm representation directly influence the order of assembly sequence. This is because the assembly sequence in MODPSO directly uses the algorithm encoding with a repair mechanism (i.e. topological sort) to ensure the solution feasibility as presented in Section 6.3.4.1. To allow the discrete encoding to be used in MODPSO, the discrete updating procedures (Section 6.3.4.4) play a vital role to retain discrete representation in the right form for the next iteration.

The GA, which is also implemented with discrete encoding in this work, experiences abrupt changes in the sequence of assembly because of standard two-point crossover and swap mutation operators. This feature increases the variability of solutions and avoids trapping in local optimum. However, it does not allow fine tuning of solutions especially in ASP and ALB, where small changes may lead to sudden improvement in results.

The discrete updating procedure in MODPSO is designed to enable fine tuning towards the end of iterations. In PSO, all particles move towards personal and global best solutions. According to discrete updating procedure (Subtraction operator (X_i-X_j)) in Section 6.3.4.4, zero velocity is given when similar element in X_i and X_j is found (this is the case when all particles move towards the best solution at the end of iterations). When majority of velocity elements are zero, only small changes occur in assembly sequence as presented by Addition operator (X_i+V_i) in Section 6.3.4.4. This feature allows fine tuning of the assembly sequences in MODPSO.

The number of different feasible assembly sequences that are generated in MODPSO is larger than standard PSO and ACO because of the sensitivity of

discrete encoding. This gives a better chance for MODPSO to explore solutions across the graph stages. Besides this, the fine tuning feature in discrete updating procedure allows MODPSO to make small changes in assembly sequence. This is very useful to test parallel (similar stage) and serial (different stage) task selection that come out with better fitness.

9.2 Main Contributions

The general contribution of this research as outlined in the research aim is the establishment of a methodology and algorithm for integrating ASP and ALB optimisation using MODPSO. In order to achieve the research aim, this research has delivered a number of main contributions to knowledge. The research contributions are listed below:

- **Integrated ASP and ALB task-based representation scheme**: The proposed integrated ASP and ALB representation scheme developed based on the assembly task is very important because it is more closely linked to real-world assembly. The proposed representation scheme clearly defines the assembly direction based on assembly tasks, which has not been done before. This contribution is significant because without clear definition of assembly direction on representation, the implementation of this scheme to real-world problems would be impossible.

- **Tuneable test problem generator**: The proposed test problem generator for integrated ASP and ALB is able to generate an unlimited number of test problems with controllable difficulty levels. This is important to supply sufficient test problems to test algorithm performance throughout all difficulty levels. The proposed test problem generator not only benefits the researchers in integrated optimisation, but is also applicable for researchers only studying ASP or ALB. This contribution is significant since there are hundreds of optimisation works in this area and this number is increasing from year to year.

- **Multi-Objective Discrete Particle Swarm Optimisation algorithm**: The proposed MODPSO algorithm is developed to optimise integrated multi-objective ASP and ALB problems. The discrete velocity and position updating procedure is used since both ASP and ALB are discrete problems. This algorithm has been tested using a different range of problem difficulties which are generated from the tuneable test problem generator and have performed well in terms of solution quality towards Pareto optimal solutions. The MODPSO's performance has been validated using artificial problems from the literature and also real-world problems.

- **Integrated mixed-model ASP and ALB**: This research has initiated the optimisation of integrated mixed-model ASP and ALB. The integrated mixed-model ASP and ALB problem is significant because in the real-world, the mixed-model assembly line is widely used in various industries to produce a wide variety of products. Besides the contribution of diversifying the new problem, the formulation of integrated mixed-model ASP and ALB allows manufacturers to gain benefits of mixed-model (e.g. cost saving by sharing assembly line) and benefits of integrated optimisation (e.g. better solution quality) together at one time.

- **Numerical comparison of integrated and sequential optimisation approaches**: Although many benefits of integrated ASP and ALB optimisation are discussed in the literature, no existing work has published the numerical comparison between sequential and integrated optimisation to justify the mentioned benefits. In this thesis, a numerical comparison between integrated and sequential optimisation approaches has been conducted to validate the benefits of better solution quality towards Pareto optimal using an integrated optimisation approach. The numerical comparison of integrated and sequential optimisation approaches gives a better understanding to use an appropriate optimisation approach for a particular problem category.

9.3 Limitations of the Research

This section discusses the limitations of the research as observed throughout the research activities. The research limitations are summarised below.

- The process of establishing a liaison matrix and precedence relation between one task and another might be difficult for a problem with a large number of tasks. This process consumes a lot of time and requires a high level of focus to avoid any mistakes during this stage which might affect the result accuracy.
- The comparison algorithms used in this research are selected based on popularity either from citation or frequency in different categories. The comparison algorithms selected are from the three most dominant algorithms in ASP and ALB, i.e. GA, ACO and PSO. It does not cover all the available algorithms for ASP and ALB because there is limited information and time to consider all of them.
- In both integrated single-model and mixed-model ASP and ALB experiments, the MODPSO algorithm performs better in finding more non-dominated solutions in Pareto optimal and exploring the search space. However, the MODPSO has limited performance in terms of uniformity of solutions presented by the *Spacing* indicator. It only shows significant improvements compared to certain algorithms in single-model and no significant improvements over any comparable algorithm in mixed-model problems.
- The artificial and real-world problems used for validation are categorised according to the size of the problem. Although the problem difficulty as presented in Section 5.2.1 (Table 5.1) considers four different inputs to differentiate the level, only the number of tasks is used for problem selection because of limited integrated ASP and ALB problems from the literature.
- Some of the assembly data in the problems used for validation are randomly generated because of limited integrated ASP and ALB problems from the literature. Whilst in the real-world problems validation, some of the assembly time data is estimated through simulated actions.

9.4 Future Research

Based on the limitations of this research, several recommendations for future research are proposed. The future directions of the research are summarised as follows.

- Automate the assembly liaison and precedence constraint establishment by implementing the geometrical data extraction from the Computer-Aided Design (CAD) model. The geometrical data manipulation has been implemented in much ASP research, but has not been extended to ALB research.
- Compare the performance of MODPSO with other algorithms that have good potential such as Simulated Annealing, Memetic Algorithm or hybrid algorithms instead of GA, ACO and PSO-based algorithms as used in this work.
- Improve the MODPSO algorithm to have better solution uniformity instead of better number of non-dominated solutions and better exploration in search space than achieved in this research.

9.5 Research Conclusions

This section concludes the research achievements in relation to the research aim and objectives as stated in Chapter 3. The following discussion compares the research objectives and achievements throughout this research.

- *Objective 1: To establish an integrated representation scheme for ASP and ALB*. This objective is achieved by proposing the integrated representation scheme for ASP and ALB that was built based on the assembly task. The detail of the representation scheme including assembly task definition, precedence constraint establishment and assembly data description have been successfully proposed and discussed in Chapter 4. The practicality of the representation scheme is proved through an example of application in Section 4.3 and real-world problem establishment in Section 8.3.

- *Objective 2: To develop a tuneable test problem generator to generate integrated ASP and ALB test problems*. This objective is accomplished by developing a test problem generator with tuneable difficulty levels. The integrated ASP and ALB test problem difficulty is controlled via four tuneable inputs (i.e. number of task, Order Strength, Time Variability ratio and Frequency Ratio) as presented in Chapter 5. The capability of the developed tuneable test problem generator to generate integrated ASP and ALB test problems in different difficulties is confirmed through a discussion of experiments in Section 5.5. The accomplishment of this objective has filled the lack of integrated ASP and ALB test problems gap.

- *Objective 3: To develop a multi-objective algorithm to optimise integrated ASP and ALB using Particle Swarm Optimisation*. This objective is completed by developing the Multi-Objective Discrete Particle Swarm Optimisation (MODPSO) algorithm in Chapter 6. The MODPSO algorithm adopts the Pareto-based approach for multi-objective optimisation and uses the discrete updating procedure to suit the discrete features of ASP and ALB problems. The proposed MODPSO algorithm is thoroughly tested at different difficulty levels using integrated ASP and ALB problems from the test problem generator. The completion of this objective has filled the gap of PSO application in integrated ASP and ALB, instead only GA in existing works.

- *Objective 4: To extend the developed algorithm to optimise integrated mixed-model ASP and ALB*. This objective is accomplished by formulating the integrated mixed-model ASP and ALB problems. The integrated mixed-model problem is formulated by transforming the precedence diagram into a joint precedence diagram which represents all the models. The MODPSO algorithm is also successfully applied to optimise this problem, as presented in Chapter 7. The achievement of this objective has filled the gap of the

integrated ASP and ALB optimisation for the generalised type of assembly problems, instead of single-model problem in existing works.

- *Objective 5: To validate the performance of optimisation algorithm through test problems from literature and real-world problems*. This objective is achieved as presented in Chapter 8. In the first part, validation is performed using artificial problems from the literature. The result of MODPSO algorithm is compared to results that are presented in original sources and also compared to results from comparable algorithms. In the second part, validation is made using real-world problems by comparing the performance of MODPSO and comparable algorithms. The results from both parts validate the performance of MODPSO to optimise integrated ASP and ALB problems.

Based on the accomplishment of research objectives, it can be concluded that this research has established a methodology and algorithm for integrating ASP and ALB optimisation using Particle Swarm Optimisation. Therefore, the aim of this research has successfully been achieved.

In conclusion, this research has demonstrated the following:

- ❖ The tuneable Test Problem Generator is capable of generating integrated ASP and ALB test problems with different levels of difficulties by using a combination of tuneable inputs. This has enabled the proposed optimisation algorithm to be tested comprehensively.

- ❖ The proposed MODPSO algorithm has shown significant improvements compared to existing algorithms in finding Pareto optimal solutions for integrated ASP and ALB problems with both single-model and mixed-models.

- ❖ The proposed MODPSO algorithm is able to solve a range of real-world problems with improved results from presented in the literature.

- In comparison with a sequential optimisation approach, the integrated ASP and ALB optimisation approach has equivalent performance for ASP objectives, while for ALB objectives, the integrated approach has better performance when an appropriate running length is used.

REFERENCES

Abraham, A., Liu, H., Grosan, C. and Xhafa, F. (2008), "Nature inspired meta-heuristics for grid scheduling: single and multi-objective optimization approaches" *Metaheuristics for Scheduling in Distributed Computing Environments*, vol. 146, pp. 247-272*,* Springer.

Akgündüz, O. S. and Tunali, S. (2011), "A review of the current applications of genetic algorithms in mixed-model assembly line sequencing", *International Journal of Production Research,* vol. 49, no. 15, pp. 4483-4503.

Akyol, S. D. and Mirac Bayhan, G. (2011), "A particle swarm optimization algorithm for maximizing production rate and workload smoothness", *Proceedings of the 2011 3rd World Congress on Nature and Biologically Inspired Computing, NaBIC 2011,* pp. 44.

Amin, M. A. and Karim, M. A. (2013), "A time-based quantitative approach for selecting lean strategies for manufacturing organisations", *International Journal of Production Research,* vol. 51, no. 4, pp. 1146-1167.

Andrés, C., Miralles, C. and Pastor, R. (2008), "Balancing and scheduling tasks in assembly lines with sequence-dependent setup times", *European Journal of Operational Research,* vol. 187, no. 3, pp. 1212-1223.

Bai, Y. W., Chen, Z. N., Bin, H. Z. and Hun, J. (2005), "An effective integration approach toward assembly sequence planning and evaluation", *International Journal of Advanced Manufacturing Technology,* vol. 27, no. 1-2, pp. 96-105.

Battini, D., Faccio, M., Ferrari, E., Persona, A. and Sgarbossa, F. (2007), "Design configuration for a mixed-model assembly system in case of low product demand", *International Journal of Advanced Manufacturing Technology,* vol. 34, no. 1-2, pp. 188-200.

Bautista, J. and Pereira, J. (2007), "Ant algorithms for a time and space constrained assembly line balancing problem", *European Journal of Operational Research,* vol. 177, no. 3, pp. 2016-2032.

Baybars, I. (1986), "Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem", *Management Science,* vol. 32, no. 8, pp. 909-932.

Becker, C. and Scholl, A. (2006), "A survey on problems and methods in generalized assembly line balancing", *European Journal of Operational Research,* vol. 168, no. 3, pp. 694-715.

Betancourt L.C. (2007), *ASALBP: the Alternative Subgraphs Assembly Line Balancing Problem: Formalization and Resolution Procedures*, PhD thesis, Technical University of Catalonia.

Bhattacharjee, T. K. and Sahu, S. (1990), "Complexity of single model assembly line balancing problems", *Engineering Costs and Production Economics,* vol. 18, no. 3, pp. 203-214.

Biswal, B. B., Choudhury, B. B., Mishra, D. and Dash, P. (2010), "An overview and comparison of four sequence generating methods for robotic assembly", *International Journal of Manufacturing Technology and Management,* vol. 20, no. 1-4, pp. 169-196.

Blum, C., Bautista, J. and Pereira, J., (2006), "Beam-ACO applied to assembly line balancing",*Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),* vol. 4150, pp. 96-107.

Blum, C., Bautista, J. and Pereira, J., (2008), "An extended beam-ACO approach to the time and space constrained simple assembly line balancing problem",*Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),* vol. 4972, pp. 85-96.

Bourjault, A. (1984), *A methodological approach of automated assembly: Development of automatic sequences operations*, PhD thesis, University of Franche-Comte.

Boysen, N., Fliedner, M. and Scholl, A. (2007), "A classification of assembly line balancing problems", *European Journal of Operational Research,* vol. 183, no. 2, pp. 674-693.

Cakir, B., Altiparmak, F. and Dengiz, B. (2011), "Multi-objective optimization of a stochastic assembly line balancing: A hybrid simulated annealing algorithm", *Computers & Industrial Engineering,* vol. 60, no. 3, pp. 376-384.

Cao, P. and Xiao, R. (2007), "Assembly planning using a novel immune approach", *International Journal of Advanced Manufacturing Technology,* vol. 31, no. 7-8, pp. 770-782.

Cao, Z. and Ma, S. (2008), "Balancing and sequencing optimization of the mixed model assembly lines", *Proceedings - International Symposium on Computer Science and Computational Technology, ISCSCT 2008,* vol. 1, pp. 732.

Capacho, L. and Pastor, R., (2006), "The ASALB problem with processing alternatives involving different tasks: Definition, formalization and resolution",*Lecture Notes in Computer Science,* vol. 3982, pp.554-563.

Chang, C. C., Tseng, H. E. and Meng, L. P. (2009), "Artificial immune systems for assembly sequence planning exploration", *Engineering Applications of Artificial Intelligence,* vol. 22, no. 8, pp. 1218-1232.

Chehade, H., Yalaoui, F. and Amodeo, L. (2013), "New multiobjective optimisation algorithms for assembly lines design", *International Journal of Advanced Operations Management,* vol. 5, no. 2, pp. 94-120.

Chen, G., Zhou, J., Cai, W., Lai, X., Lin, Z. and Menassa, R. (2006), "A framework for an automotive body assembly process design system", *CAD Computer Aided Design,* vol. 38, no. 5, pp. 531-539.

Chen, J. C., Chen, C., Su, L., Wu, H. and Sun, C. (2012), "Assembly line balancing in garment industry", *Expert Systems with Applications,* vol. 39, no. 11, pp. 10073-10081.

Chen, R., Lu, K. and Yu, S. (2002), "A hybrid genetic algorithm approach on multi-objective of assembly planning problem", *Engineering Applications of Artificial Intelligence,* vol. 15, no. 5, pp. 447-457.

Chen, S. and Liu, Y. (2001), "An adaptive genetic assembly-sequence planner", *International Journal of Computer Integrated Manufacturing,* vol. 14, no. 5, pp. 489-500.

Chen, W., Tai, P., Deng, W. and Hsieh, L. (2008), "A three-stage integrated approach for assembly sequence planning using neural networks", *Expert Systems with Applications,* vol. 34, no. 3, pp. 1777-1786.

Chica, M., Cordón, O., Damas, S. and Bautista, J. (2010), "Multiobjective constructive heuristics for the 1/3 variant of the time and space assembly line balancing problem: ACO and random greedy search", *Information Sciences,* vol. 180, no. 18, pp. 3465-3487.

Chica, M., Cordón, O., Damas, S. and Bautista, J. (2011), "Including different kinds of preferences in a multi-objective ant algorithm for time and space assembly line balancing on different Nissan scenarios", *Expert Systems with Applications,* vol. 38, no. 1, pp. 709-720.

Chica, M., Cordón, Ó., Damas, S., Pereira, J. and Bautista, J. (2008), "Incorporating preferences to a multi-objective ant colony algorithm for time and space assembly line balancing", *Lecture Notes in Computer Science,* vol. 5217, pp. 331-338.

Choi, Y., Lee, D. M. and Cho, Y. B. (2009), "An approach to multi-criteria assembly sequence planning using genetic algorithms", *International Journal of Advanced Manufacturing Technology,* vol. 42, no. 1-2, pp. 180-188.

Chutima, P. and Chimklai, P. (2012), "Multi-objective two-sided mixed-model assembly line balancing using particle swarm optimisation with negative knowledge", *Computers and Industrial Engineering,* vol. 62, no. 1, pp. 39-55.

Coello Coello, C. A. and Lechuga, M. S. (2002), "MOPSO: a proposal for multiple objective particle swarm optimization", *Proceedings of the 2002 Congress on Evolutionary Computation, 2002, CEC '02,* vol. 2, pp. 1051.

Coolidge, F. (2000), *Statistics: A Gentle Introduction,* SAGE Publication Ltd, London.

Corallo, A., Margherita, M. and Pascali, G. (2010), "Digital Mock-up to Optimize the Assembly of a Ship Fuel System", *Journal of Modelling and Simulation of Systems,* vol. 1, no. 1, pp. 4-12.

De Fazio, T. L. and Whitney, D. E. (1987), "Simplified generation of all mechanical assembly sequences", *IEEE Journal of Robotics and Automation,* vol. 3, no. 6, pp. 640-658.

De Lit, P., Latinne, P., Rekiek, B. and Delchambre, A. (2001), "Assembly planning with an ordering genetic algorithm", *International Journal of Production Research,* vol. 39, no. 16, pp. 3623-3640.

Deb K. (2001), *Multi-Objective Optimization using Evolutionary Algorithm,* John Wiley & Sons Inc., England.

Demoly, F., Yan, X., Eynard, B., Gomes, S. and Kiritsis, D. (2012), "Integrated product relationships management: A model to enable concurrent product design and assembly sequence planning", *Journal of Engineering Design,* vol. 23, no. 7, pp. 544-561.

Demoly, F., Yan, X., Eynard, B., Rivest, L. and Gomes, S. (2011), "An assembly oriented design framework for product structure engineering and assembly sequence planning", *Robotics and Computer-Integrated Manufacturing,* vol. 27, no. 1, pp. 33-46.

Elbeltagi, E., Hegazy, T. and Grierson, D. (2005), "Comparison among five evolutionary-based optimization algorithms", *Advanced Engineering Informatics,* vol. 19, no. 1, pp. 43-53.

Fokkert, J. and Kok, T. G. (1997), "The mixed and multi model line balancing problem: a comparison", *European Journal of Operational Research,* vol. 100, pp. 399-412.

Fouda, P., De Lit, P., Rekiek, B. and Delchambre, A. (2001), "Generation of precedence graphs for a product family using a disassembly approach",

Proceedings of the IEEE International Symposium on Assembly and Task Planning, pp. 226-231.

Gao, J., Sun, L., Wang, L. and Gen, M. (2009), "An efficient approach for type II robotic assembly line balancing problems", *Computers and Industrial Engineering,* vol. 56, no. 3, pp. 1065-1080.

Gao, L., Qian, W., Li, X. and Wang, J. (2010), "Application of memetic algorithm in assembly sequence planning", *International Journal of Advanced Manufacturing Technology,* vol. 49, no. 9-12, pp. 1175-1184.

Goldberg, D. (2007), *Genetic algorithms: the design of innovation,* Springer Berlin Heidelberg, New York.

Goldwasser, M., and Motwani, R. (1997), "Intractability of Assembly Sequencing: Unit Disks in the Plane", *Lecture Notes in Computer Science (Springer-Verlag)*, vol. 1272, pp. 307-320.

Gonçalves, J. F. and De Almeida, J. R. (2002), "A hybrid genetic algorithm for assembly line balancing", *Journal of Heuristics,* vol. 8, no. 6, pp. 629-642.

Gottipolu, R. B. and Ghosh, K. (1997), "Representation and selection of assembly sequences in computer-aided assembly process planning", *International Journal of Production Research,* vol. 35, no. 12, pp. 3447-3465.

Gu, L., Hennequin, S., Sava, A. and Xie, X. (2007), "Assembly line balancing problems solved by estimation of distribution", *3rd IEEE International Conference on Automation Science and Engineering, IEEE CASE 2007*, pp. 123-127.

Gu, T. and Liu, H. (2008), "The symbolic OBDD scheme for generating mechanical assembly sequences", *Formal Methods in System Design,* vol. 33, no. 1-3, pp. 29-44.

Guan, Q., Liu, J. H. and Zhong, Y. F. (2002), "A concurrent hierarchical evolution approach to assembly process planning", *International Journal of Production Research,* vol. 40, no. 14, pp. 3357-3374.

Hager, T., Ahmed, M. and Faouzi, M. (2013), "Assembly Line Resource Assignment and Balancing Problem of Type 2", in *Design and Modeling of Mechanical Systems,* Springer, pp. 627-634.

Hamta, N., Fatemi Ghomi, S. M. T., Jolai, F. and Akbarpour Shirazi, M. (2013), "A hybrid PSO algorithm for a multi-objective assembly line balancing problem with flexible operation times, sequence-dependent setup times and learning effect", *International Journal of Production Economics,* vol. 141, no. 1, pp. 99-111.

Hamza, R. M. A., and Al-Manaa, J. Y. (2013), “Selection of Balancing Method for Manual Assembly Line of Two Stages Gearbox”, *Global Perspectives on Engineering Management*, vol. 2(2), pp. 70-81.

Haq, A. N., Rengarajan, K. and Jayaprakash, J. (2006), "A hybrid genetic algorithm approach to mixed-model assembly line balancing", *International Journal of Advanced Manufacturing Technology,* vol. 28, no. 3-4, pp. 337-341.

Homem de Mello, L. S. and Sanderson, A. C. (1990), "AND/OR graph representation of assembly plans", *IEEE Transactions on Robotics and Automation,* vol. 6, no. 2, pp. 188-199.

Hu, S. J., Zhu, X., Wang, H. and Koren, Y. (2008), "Product variety and manufacturing complexity in assembly systems and supply chains", *CIRP Annals - Manufacturing Technology,* vol. 57, no. 1, pp. 45-48.

Hui, C., Yuan, L. and Kai-Fu, Z. (2009), "Efficient method of assembly sequence planning based on GAAA and optimizing by assembly path feedback for complex product", *International Journal of Advanced Manufacturing Technology,* vol. 42, no. 11-12, pp. 1187-1204.

Jeong, S., Hasegawa, S., Shimoyama, K. and Obayashi, S. (2009), "Development and investigation of efficient GA/PSO-hybrid algorithm applicable to real-world design optimization", *Computational Intelligence Magazine, IEEE,* vol. 4, no. 3, pp. 36-44.

Jianping, D., Chun, S. and Jun, L. (2011), "A Discrete Particle Swarm Optimization Algorithm for Assembly Line Balancing Problem of Type 1", *IEEEThird International Conference on Measuring Technology and Mechatronics Automation (ICMTMA),* vol. 1, pp. 44.

Johnson, R. (1981), "Assembly line balancing algorithms: computation comparisons", *International Journal of Production Research,* vol. 19, no. 3, pp. 277-287.

Jonnalagedda, V. L. V. and Dabade, B. M. (2010), "Heuristic procedure for mixed model assembly line balancing problem", *IEEE International Conference on Industrial Engineering and Engineering Management,* pp. 552.

Kara, Y., Özgüven, C., Seçme, N. Y. and Chang, C. (2011), "Multi-objective approaches to balance mixed-model assembly lines for model mixes having precedence conflicts and duplicable common tasks", *International Journal of Advanced Manufacturing Technology,* vol. 52, no. 5-8, pp. 725-737.

Kennedy, J. and Eberhart, R. (1995), "Particle swarm optimization", *IEEE International Conference on Neural Networks*, vol. 4, pp. 1942-1948.

Khoo, L. P. and Alisantoso, D. (2003), "Line balancing of PCB assembly line using immune algorithms", *Engineering with Computers,* vol. 19, no. 2-3, pp. 92-100.

Kilbridge, M. and Wester, L. (1961), "The Balance Delay Problem", *Management Science,* vol. 8, no. 1, pp. 69-84.

Kilincci, O. (2010), "A Petri net-based heuristic for simple assembly line balancing problem of type 2", *International Journal of Advanced Manufacturing Technology,* vol. 46, no. 1-4, pp. 329-338.

Kilincci, O. and Bayhan, G. M. (2006), "A Petri net approach for simple assembly line balancing problems", *International Journal of Advanced Manufacturing Technology,* vol. 30, no. 11-12, pp. 1165-1173.

Kilincci, O. and Bayhan, G. M. (2008), "A P-invariant-based algorithm for simple assembly line balancing problem of type-1", *International Journal of Advanced Manufacturing Technology,* vol. 37, no. 3-4, pp. 400-409.

Kim, Y. K., Kim, J. Y. and Kim, Y. (2000), "A Coevolutionary Algorithm for Balancing and Sequencing in Mixed Model Assembly Lines", *Applied Intelligence,* vol. 13, no. 3, pp. 247-258.

Koç, A. (2005), *Inquiring the Main Assumption of the Assembly Line Balancing Problem: Solution Procedures Using AND/OR Graph*, MSc thesis, Bilkent University, Ankara.

Kumar, E. and Annamalai, K. (2012), "Optimal assembly sequence planning of fixture in a virtual environment", *International Journal of Engineering Sciences Research,* vol. 3, no. 6, pp. 860-865.

Lai, H. and Huang, C. (2003), "Integrated assembly plan generation system for grouped product families", *International Journal of Production Research,* vol. 41, no. 17, pp. 4041-4061.

Lai, H. and Huang, C. (2004), "A systematic approach for automatic assembly sequence plan generation", *International Journal of Advanced Manufacturing Technology,* vol. 24, no. 9-10, pp. 752-763.

Lapierre, S. D., Ruiz, A. and Soriano, P. (2006), "Balancing assembly lines with tabu search", *European Journal of Operational Research,* vol. 168, no. 3, pp. 826-837.

Lazzerini, B. and Marcelloni, F. (2000), "Genetic algorithm for generating optimal assembly plans", *Artificial Intelligence in Engineering,* vol. 14, no. 4, pp. 319-329.

Lendak, I., Erdeljan, A., Čapko, D. and Vukmirović, S. (2010), "Algorithms in electric power system one-line diagram creation: The soft computing approach", *IEEE International Conference on Systems, Man and Cybernetics*, pp. 2867-2873.

Levitin, G., Rubinovitz, J. and Shnits, B. (2006), "A genetic algorithm for robotic assembly line balancing", *European Journal of Operational Research,* vol. 168, no. 3, pp. 811-825.

Li, M., Wu, B., Hu, Y., Jin, C. and Shi, T. (2013a), "A hybrid assembly sequence planning approach based on discrete particle swarm optimization and evolutionary direction operation", *The International Journal of Advanced Manufacturing Technology,* Article in Press.

Li, R., Chen, H. and Xie, L. (2013b), "Assembly sequence planning including tool check based on improved ant-colony algorithm", *Advanced Materials Research,* vol. 670, pp. 3-9.

Li, J. R., Khoo, L. P. and Tor, S. B. (2003), "A Tabu-enhanced genetic algorithm approach for assembly process planning", *Journal of Intelligent Manufacturing,* vol. 14, no. 2, pp. 197-208.

Li, S. and Shan, H. (2008), "GSSA and ACO for assembly sequence planning: A comparative study", *Proceedings of the IEEE International Conference on Automation and Logistics*, pp. 1270-1275.

Lin, S., Xun, T. and Bai, R. (2003), "Double guided genetic algorithm for sequence planning", *National Conference in Engineering Computation,* 13-15 August 2003, Wuhan, China, pp. 419-423.

Lin, Y. Y., Che, Z. H., Chiang, T., Che, Z. and Chiang, C. J. (2009), "A Bi-objective model for concurrent planning of supplier selection and assembly sequence planning", *Global Perspective for Competitive Enterprise, Economy and Ecology,* Springer London, pp. 573-580.

Liu, C. Y. and Wen, H. J. (2013), "Application of multi-objective culture particle swarm optimization in complex product assembly line balancing", *Advanced Materials Research,* vol. 694, pp. 3526-3530.

Liu, W. P., Wang, M. and Zhu, Z. J. (2013), "Bottleneck mitigation of assembly lines with memetic algorithms", *Advanced Materials Research,* vol. 655, pp. 1631-1635.

Liu, J., Wang, Y. and Gu, Z. (2009), "Generation of optimal assembly sequences using particle swarm optimization", *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 5, pp. 11-18.

Liu, S. B., Ong, H. L. and Huang, H. C. (2003), "Two bi-directional heuristics for the assembly line type II problem", *International Journal of Advanced Manufacturing Technology,* vol. 22, no. 9-10, pp. 656-661.

Lu, C., Wong, Y. S. and Fuh, J. Y. H. (2006), "An enhanced assembly planning approach using a multi-objective genetic algorithm", *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture,* vol. 220, no. 2, pp. 255-272.

Lu, J., Zhang, L., Yang, H. and Du, J. (2010), "Improved strategy of particle swarm optimisation algorithm for reactive power optimization", *International Journal of Bio-Inspired Computation,* vol. 2, no. 1, pp. 27-33.

Luke, S. (2010), *Essentials of Metaheuristics,* 1st Edition, Lulu, California, USA.

Lv, H. and Lu, C. (2009), "A discrete particle swarm optimization algorithm for assembly sequence planning", *Proceedings of 8th International Conference on Reliability, Maintainability and Safety*, pp. 1119-1122.

Lv, H. and Lu, C. (2010), "An assembly sequence planning approach with a discrete particle swarm optimization algorithm", *International Journal of Advanced Manufacturing Technology,* vol. 50, no. 5-8, pp. 761-770.

Lv, H. G., Lu, C. and Zha, J. (2010), "A hybrid DPSO-SA approach to assembly sequence planning", *IEEE International Conference on Mechatronics and Automation*, no. 5589203, pp. 1998-2003.

Lv, Q. (2011), "Simple assembly line balancing using particle swarm optimization algorithm", *International Journal of Digital Content Technology and its Applications,* vol. 5, no. 6, pp. 297-304.

Marian, R. M. (2003), *Optimisation of Assembly Sequences using Genetic Algorithm*, PhD thesis, University of South Australia, ProQuest.

Marian, R. M., Luong, L. H. S. and Abhary, K. (2006), "A genetic algorithm for the optimisation of assembly sequences", *Computers and Industrial Engineering,* vol. 50, no. 4, pp. 503-527.

Martino, L. and Pastor, R. (2010), "Heuristic procedures for solving the general assembly line balancing problem with setups", *International Journal of Production Research,* vol. 48, no. 6, pp. 1787-1804.

Mastor, A. (1970), "An experimental investigation and comparative evaluation of production line balancing techniques", *Management Science,* vol. 16, no. 11, pp. 728-746.

McMullen, P. R. and Tarasewich, P. (2003), "Using ant techniques to solve the assembly line balancing problem", *IIE Transactions (Institute of Industrial Engineers),* vol. 35, no. 7, pp. 605-617.

McMullen, P. R. and Tarasewich, P. (2006), "Multi-objective assembly line balancing via a modified ant colony optimization technique", *International Journal of Production Research,* vol. 44, no. 1, pp. 27-42.

Mingxing, D., Qiuhua, T. and Zhe, L. (2011), "An improved assembly sequence planning approach using ant colony algorithm", *International Review on Computers and Software,* vol. 6, no. 7, pp. 1307-1312.

Mitrović-Minić, S. and Krishnamurti, R. (2006), "The multiple TSP with time windows: Vehicle bounds based on precedence graphs", *Operations Research Letters,* vol. 34, no. 1, pp. 111-120.

Mohd, A. and Azmi, W. (2010), "Productivity improvement using improved genetic programming", *Symposium of Postgraduate Studies,* 16-18 Sept 2010, Kuala Lumpur, pp. 77-82.

Moon, C., Kim, J., Choi, G. and Seo, Y. (2002), "An efficient genetic algorithm for the traveling salesman problem with precedence constraints", *European Journal of Operational Research,* vol. 140, no. 3, pp. 606-617.

Moon, I., Logendran, R. and Lee, J. (2009), "Integrated assembly line balancing with resource restrictions", *International Journal of Production Research,* vol. 47, no. 19, pp. 5525-5541.

Mozdgir, A., Mahdavi, I., Badeleh, I. S. and Solimanpur, M. (2013), "Using the Taguchi method to optimize the differential evolution algorithm parameters for minimizing the workload smoothness index in simple assembly line balancing", *Mathematical and Computer Modelling,* vol. 57, no. 1–2, pp. 137-151.

Mukred, J. A. A., Ibrahim, Z., Ibrahim, I., Adam, A., Wan, K., Md. Yusof, Z. and Mokhtar, N. (2012), "A Binary Particle Swarm Optimization Approach to Optimize assembly sequence planning", *Advanced Science Letters,* vol. 13, pp. 732-738.

Mukund Nilakantan, J. and Ponnambalam, S. G. (2012), "An efficient PSO for type II robotic assembly line balancing problem", *IEEE International Conference on Automation Science and Engineering,* pp. 600-605.

Muslim, M. T., Selamat, H. and Mukred, J. A. (2013), "Optimizing assembly sequence time using Particle Swarm Optimization (PSO)", *Applied Mechanics and Materials,* vol. 315, pp. 88-92.

Mutlu, Ö., Polat, O. and Supciller, A. A. (2013), "An iterative genetic algorithm for the assembly line worker assignment and balancing problem of type-II", *Computers & Operations Research,* vol. 40, no. 1, pp. 418-426.

Nearchou, A. C. (2008), "Multi-objective balancing of assembly lines by population heuristics", *International Journal of Production Research,* vol. 46, no. 8, pp. 2275-2297.

Nearchou, A. C. (2011), "Maximizing production rate and workload smoothing in assembly lines using particle swarm optimization", *International Journal of Production Economics,* vol. 129, no. 2, pp. 242-250.

Nof, S. Y., Wilhelm W.E. and Warnecke, H. (1997), *Industrial Assembly,* Chapman & Hall, London.

Otto, A., Otto, C. and Scholl, A. (2011), "SALBPGen – A systematic data generator for (simple) assembly line balancing", *Jena Research Papers in Business and Economics,* [Online], vol. 5, available at: pubdb.wiwi.uni-jena.de/pdf/wp-jbe201105.pdf.

Özcan, U. and Toklu, B. (2009), "A new hybrid improvement heuristic approach to simple straight and U-type assembly line balancing problems", *Journal of Intelligent Manufacturing,* vol. 20, no. 1, pp. 123-136.

Padrón, M., María de los, A. I., Resto, P. and Mejía, H. P. (2009), "A methodology for cost-oriented assembly line balancing problems", *Journal of Manufacturing Technology Management,* vol. 20, no. 8, pp. 1147-1165.

Pan C (2005), *Integrating CAD files and automatic assembly sequence planning*, PhD thesis, Iowa State University, ProQuest.

Pan C., Smith S.F. and Smith G.C. (2006), "Automatic assembly sequence planning from STEP CAD files", *International Journal of Computer Integrated Manufacturing,* vol. 19, no. 8, pp. 775-783.

Panneton, F., L'Ecuyer, P. and Matsumoto, M. (2006), "Improved long-period generators based on linear recurrences modulo 2", *ACM Transactions on Mathematical Software,* vol. 32, no. 1, pp. 1-16.

Pasupuleti, S. and Battiti, R. (2006), "The gregarious particle swarm optimizer (G-PSO)", *GECCO 2006 - Genetic and Evolutionary Computation Conference,* vol. 1, pp. 67-74.

Perween, S., Zaheer, A. and Khalid, R. (2013), "Classification and balancing of an automotive assembly line", *International Asia Conference on Industrial Engineering and Management Innovation (IEMI2012) Proceedings,* Springer, pp. 429-438.

Petropoulos, D. I. and Nearchou, A. C. (2011), "A particle swarm optimization algorithm for balancing assembly lines", *Assembly Automation,* vol. 31, no. 2, pp. 118-129.

Ponnambalam, S. G., Aravindan, P. and Naidu, G. M. (1999), “A comparative evaluation of assembly line balancing heuristics”, *The International Journal of Advanced Manufacturing Technology*, vol. 15(8), pp. 577-586.

Ponnambalam, S. G., Aravindan, P. and Naidu, G. M. (2000), "Multi-objective genetic algorithm for solving assembly line balancing problem", *International Journal of Advanced Manufacturing Technology,* vol. 16, no. 5, pp. 341-352.

Premalatha, K. and Natarajan, A. (2009), "Hybrid PSO and GA for global maximization", *International Journal of Open Problem in Computer Science and Mathematics,* vol. 2, no. 4, pp. 597-608.

Qian, W., Yan, Y., Zhao, X. and Pagello, E. (1996), "List representation of assembly sequences", *Proceedings of the IEEE International Conference on Industrial Technology*, pp. 315-319.

Qin, Y. and Xu, Z. (2007), "Assembly process planning using a multi-objective optimization method", *Proceedings of the 2007 IEEE International Conference on Mechatronics and Automation*, 4303610, pp. 593-598.

Rahmat-Samii, Y. (2003), "Genetic algorithm (GA) and particle swarm optimization (PSO) in engineering electromagnetics", *17th International Conference on Applied Electromagnetics and Communications,* IEEE, pp. 1-5.

Raj, M. V., Sankar, S. S. and Ponnambalam, S. (2012), "Particle swarm optimization algorithm to maximize assembly efficiency", *The International Journal of Advanced Manufacturing Technology,* vol. 59, no. 5-8, pp. 719-736.

Rameshkumar, K., Suresh, R. K. and Mohanasundaram, K. M. (2005), "Discrete particle swarm optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makespan", *Lecture Notes in Computer Science,* vol. 3612, pp. 572.

Rardin, R. L. and Uzsoy, R. (2001), "Experimental evaluation of heuristic optimization algorithms: A tutorial", *Journal of Heuristics,* vol. 7, no. 3, pp. 261-304.

Rashid, M. F. F., Hutabarat, W. and Tiwari, A. (2012a), "Development of tuneable test problem generator for assembly sequence planning and assembly line balancing", *Proceedings of the Institution of Mechanical*

Engineers, Part B: Journal of Engineering Manufacture, vol. 226, no. 11, pp. 1900-1913.

Rashid, M. F. F., Hutabarat, W. and Tiwari, A. (2012b), "A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches", *International Journal of Advanced Manufacturing Technology,* vol. 59, no. 1-4, pp. 335-349.

Rashid, M. F. F., Tiwari, A. and Hutabarat, W. (2011), "An Integrated Representation Scheme for Assembly Sequence Planning and Assembly Line Balancing", *Proceedings of the 9th International Conference of Manufacturing Research, ICMR 2011,* 6-8 September 2011, Glasgow, UK, pp. 125-131.

Razali, N. M. and Geraghty, J. (2011), "Biologically inspired genetic algorithm to minimize idle time of the assembly line balancing", *Proceedings of the 2011 3rd World Congress on Nature and Biologically Inspired Computing, NaBIC 2011,* pp. 105-110.

Roshani, A., Roshani, A., Roshani, A., Salehi, M. and Esfandyari, A. (2013), "A simulated annealing algorithm for multi-manned assembly line balancing problem", *Journal of Manufacturing Systems,* vol. 32, no. 1, pp. 238-247.

Sabuncuoglu, I., Erel, E. and Tanyer, M. (2000), "Assembly line balancing using genetic algorithms", *Journal of Intelligent Manufacturing,* vol. 11, no. 3, pp. 295-310.

Salum, L. and Supciller, A.A., (2008), "Rule-Based modeling of assembly constraints for line balancing",*Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5227, pp. 783-789.

Scholl, A. (1993), "Data of Assembly Line Balancing Problem", *Schriften zur Quantitativen Betriebswirtschaftslehre 16/1993, TU Darmstadt,* vol. 16/1993.

Scholl, A. and Becker, C. (2006), "State-of-the-art exact and heuristic solution procedures for simple assembly line balancing", *European Journal of Operational Research,* vol. 168, no. 3, pp. 666-693.

Senin, N., Groppetti, R. and Wallace, D. R. (2000), "Concurrent assembly planning with genetic algorithms", *Robotics and Computer-Integrated Manufacturing,* vol. 16, no. 1, pp. 65-72.

Seyed-Alagheband, S., Ghomi, S. M. T. F. and Zandieh, M. (2011), "A simulated annealing algorithm for balancing the assembly line type II problem with sequence-dependent setup times between tasks", *International Journal of Production Research,* vol. 49, no. 3, pp. 805-825.

Shan, H., Li, S., Gong, D. and Lou, P. (2006), "Genetic simulated annealing algorithm-based assembly sequence planning", *IET Conference Publications*, vol. 524, pp. 1573-1579.

Shan, H., Zhou, S. and Sun, Z. (2009), "Research on assembly sequence planning based on genetic simulated annealing algorithm and ant colony optimization algorithm", *Assembly Automation,* vol. 29, no. 3, pp. 249-256.

Shayeghi, H., Shayanfar, H., Safari, A. and Aghmasheh, R. (2010), "A robust PSSs design using PSO in a multi-machine environment", *Energy Conversion and Management,* vol. 51, no. 4, pp. 696-702.

Shinzawa, H., Jiang, J., Iwahashi, M., Noda, I. and Ozaki, Y. (2007), "Self-modeling curve resolution (SMCR) by particle swarm optimization (PSO)", *Analytica Chimica Acta,* vol. 595, no. 1–2, pp. 275-281.

Shuang, B., Chen, J. and Li, Z. (2008), "Microrobot based micro-assembly sequence planning with hybrid ant colony algorithm", *International Journal of Advanced Manufacturing Technology,* vol. 38, no. 11-12, pp. 1227-1235.

Simaria, A. S. and Vilarinho, P. M. (2004), "A genetic algorithm based approach to the mixed-model assembly line balancing problem of type II", *Computers and Industrial Engineering,* vol. 47, no. 4, pp. 391-407.

Sinanoğlu, C. and Börklü, H. R. (2005), "An assembly sequence-planning system for mechanical parts using neural network", *Assembly Automation,* vol. 25, no. 1, pp. 38-52.

Sinavandam, S. N. and Deepa, S. N. (2008), *Introduction to Genetic Algorithms,* Springer Berlin Heidelberg, New York.

Sinha, N. and Purkayastha, B. (2004), "PSO embedded evolutionary programming technique for nonconvex economic load dispatch", *Power Systems Conference and Exposition,* IEEE, pp. 66-71.

Smith, G. C. and Smith, S. S. (2002), "An enhanced genetic algorithm for automated assembly planning", *Robotics and Computer-Integrated Manufacturing,* vol. 18, no. 5-6, pp. 355-364.

Smith, S. S. (2004), "Using multiple genetic operators to reduce premature convergence in genetic assembly planning", *Computers in Industry,* vol. 54, no. 1, pp. 35-49.

Smith, S. S. and Liu, Y. (2001), "The application of multi-level genetic algorithms in assembly planning", *Journal of Industrial Technology,* vol. 17, no. 4.

Su, P. and Lu, Y. (2007), "Combining genetic algorithm and simulation for the mixed-model assembly line balancing problem", *Proceedings - Third International Conference on Natural Computation,* vol. 4, pp. 314-318.

Su, Q. (2009), "A hierarchical approach on assembly sequence planning and optimal sequences analyzing", *Robotics and Computer-Integrated Manufacturing,* vol. 25, no. 1, pp. 224-234.

Sulaiman, M. N. I., Choo, Y. and Chong, K. E. (2011), "Ant colony optimization with look forward ant in solving assembly line balancing problem", *Conference on Data Mining and Optimization,* pp. 115-121.

Sun, Q. (2010), "Mixed-model assembly line balancing based on PSO-SA alternate algorithm", *2010 International Conference on Intelligent Computation Technology and Automation,* vol. 2, pp. 687-690.

Suwannarongsri, S., Limnararat, S. and Puangdownreong, D. (2007), "A new hybrid intelligent method for assembly line balancing", *IEEE International Conference on Industrial Engineering and Engineering Management*, 4419365,pp. 1115-1119.

Suwannarongsri, S. and Puangdownreong, D. (2008), "Multi-objective assembly line balancing via adaptive tabu search method with partial random permutation technique", *IEEE International Conference on Industrial Engineering and Engineering Management*, 4737881, pp. 312-316.

Tambe, P. (2006), *Balancing Mixed-model Assembly Line to Reduce Work Overload in a Multi-level Production* System, MSc Thesis, Louisiana State University, USA.

Tasan, S. O. and Tunali, S. (2008), "A review of the current applications of genetic algorithms in assembly line balancing", *Journal of Intelligent Manufacturing,* vol. 19, no. 1, pp. 49-69.

Tasan, S.O. and Tunali, S. (2006), "Improving the genetic algorithms performance in simple assembly line balancing",*Lecture Notes in Computer Science*, vol. 3984, pp. 78-87.

Tijo, S. and and Numar, R. (2008), "Heuristic programming for assembly line balancing", *Regional Conference of Mathematical Programming,* pp. 143-147, Seoul.

Toksari, M. D., Işleyen, S. K., Güner, E. and Baykoç, O. F. (2010), "Assembly line balancing problem with deterioration tasks and learning effect", *Expert Systems with Applications,* vol. 37, no. 2, pp. 1223-1228.

Tseng, H. (2011), "An improved ant colony system for assembly sequence planning based on connector concept", *Lecture Notes in Electrical Engineering,* 97, vol. 1, pp. 881-888.

Tseng, H., Chen, M., Chang, C. and Wang, W. (2008), "Hybrid evolutionary multi-objective algorithms for integrating assembly sequence planning and assembly line balancing", *International Journal of Production Research,* vol. 46, no. 21, pp. 5951-5977.

Tseng, H., Li, J. and Chang, Y. (2004), "Connector-based approach to assembly planning using a genetic algorithm", *International Journal of Production Research,* vol. 42, no. 11, pp. 2243-2261.

Tseng, H. and Tang, C. (2006), "A sequential consideration for assembly sequence planning and assembly line balancing using the connector concept", *International Journal of Production Research,* vol. 44, no. 1, pp. 97-116.

Tseng, H., Wang, W. and Shih, H. (2007), "Using memetic algorithms with guided local search to solve assembly sequence planning", *Expert Systems with Applications,* vol. 33, no. 2, pp. 451-467.

Tseng, Y., Chen, J. and Huang, F. (2010a), "A multi-plant assembly sequence planning model with integrated assembly sequence planning and plant assignment using GA", *International Journal of Advanced Manufacturing Technology,* vol. 48, no. 1-4, pp. 333-345.

Tseng, Y., Chen, J. and Huang, F. (2010b), "A particle swarm optimisation algorithm for multi-plant assembly sequence planning with integrated assembly sequence planning and plant assignment", *International Journal of Production Research,* vol. 48, no. 10, pp. 2765-2791.

Udeshi, T. and Tsui, K. (2005), "Assembly sequence planning for automated micro assembly", *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, pp. 98-105.

Urban, T. L. and Chiang, W. (2006), "An optimal piecewise-linear program for the U-line balancing problem with stochastic task times", *European Journal of Operational Research,* vol. 168, no. 3, pp. 771-782.

Venkatesh, J. V. L. and Dabade, B. M. (2008), "Evaluation of performance measures for representing operational objectives of a mixed model assembly line balancing problem", *International Journal of Production Research,* vol. 46, no. 22, pp. 6367-6388.

Wang, H. S., Che, Z. H. and Chiang, C. J. (2012), "A hybrid genetic algorithm for multi-objective product plan selection problem with ASP and ALB", *Expert Systems with Applications,* vol. 39, no. 5, pp. 5440-5450.

Wang, J. F., Liu, J. H. and Zhong, Y. F. (2005), "A novel ant colony algorithm for assembly sequence planning", *International Journal of Advanced Manufacturing Technology,* vol. 25, no. 11-12, pp. 1137-1143.

Wang, W. and Tseng, H. (2009), "Complexity estimation for genetic assembly sequence planning", *Journal of the Chinese Institute of Industrial Engineers,* vol. 26, no. 1, pp. 44-52.

Wang, Y. and Liu, J. H. (2010), "Chaotic particle swarm optimization for assembly sequence planning", *Robotics and Computer-Integrated Manufacturing,* vol. 26, no. 2, pp. 212-222.

Wee, T. S., and Magazine, M. J. (1982), "Assembly line balancing as generalized bin packing", *Operations Research Letters*, vol. 1(2), pp. 56-58.

Whitney, D. E. (2004), *Mechanical Assemblies: Their Design, Manufacture and Role in Product Development,* Oxford University Press, New York.

Xinchao, Z. (2010), "A perturbed particle swarm algorithm for numerical optimization", *Applied Soft Computing Journal,* vol. 10, no. 1, pp. 119-124.

Xing, Y. and Wang, Y. (2012), "Assembly sequence planning based on a hybrid particle swarm optimisation and genetic algorithm", *International Journal of Production Research,* vol. 50, no. 24, pp. 7303-7312.

Xing, Y., Wang, Y. and Zhao, X., (2010), "A particle swarm algorithm for assembly sequence planning"*, Advanced Materials Research*, vol. 3243, pp. 97-101.

Yagmahan, B. (2011), "Mixed-model assembly line balancing using a multi-objective ant colony optimization approach", *Expert Systems with Applications,* vol. 38, no. 10, pp. 12453-12461.

Yeh, D. and Kao, H. (2009), "A new bidirectional heuristic for the assembly line balancing problem", *Computers and Industrial Engineering,* vol. 57, no. 4, pp. 1155-1160.

Yolmeh, A. and Kianfar, F. (2012), "An efficient hybrid genetic algorithm to solve assembly line balancing problem with sequence-dependent setup times", *Computers and Industrial Engineering,* vol. 62, no. 4, pp. 936-945.

Yoosefelahi, A., Aminnayeri, M., Mosadegh, H. and Ardakani, H. D. (2012), "Type II robotic assembly line balancing problem: An evolution strategies algorithm for a multi-objective model", *Journal of Manufacturing Systems,* vol. 31, no. 2, pp. 139-151.

Yu, H., Yu, J. and Zhang, W., (2009), "An particle swarm optimization approach for assembly sequence planning"*, Applied Mechanics and Materials*, vol. 1228, pp. 16-19.

Yu, J. and Wang, C. (2013), "A max-min ant colony system for assembly sequence planning", *International Journal of Advanced Manufacturing Technology,* Article in Press, pp. 1-17.

Yu, J. and Yin, Y. (2010), "Assembly line balancing based on an adaptive genetic algorithm", *International Journal of Advanced Manufacturing Technology,* vol. 48, no. 1-4, pp. 347-354.

Yuan, B., Zhang, C., Lian, K. and Shao, X. (2012), "A hybrid honey-bees mating optimization algorithm for assembly sequence planning problem", *Proceedings - International Conference on Natural Computation,* pp. 1135-1140.

Zacharia, P. T. and Nearchou, A. C. (2012), "Multi-objective fuzzy assembly line balancing using genetic algorithms", *Journal of Intelligent Manufacturing,* vol. 23, no. 3, pp. 615-627.

Zeng, C., Gu, T., Zhong, Y. and Cai, G. (2011), "A multi-agent evolutionary algorithm for connector-based assembly sequence planning", *Procedia Engineering,* vol. 15, pp. 3689-3693.

Zeng, C., Gu, T., Chang, L. and Li, F. (2013), "A novel multi-agent evolutionary algorithm for assembly sequence planning", *Journal of Software,* vol. 8, no. 6, pp. 1518-1525.

Zeng, G. and Jiang, Y. (2010), "A modified PSO algorithm with line search", *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on,* IEEE, pp. 1-4.

Zha, X. F. and Du, H. (2001), "An integrated representational model for concurrent assembly design and planning", *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, 2001, 387-392.

Zhang, J., Sun, J. and He, Q. (2010), "An approach to assembly sequence planning using ant colony optimization", *2010 International Conference on Intelligent Control and Information Processing (ICICIP)*, pp. 230-233.

Zhang, J., Yang, Y. and Feng, S., (2012a), "An approach to assembly sequence planning using simulated annealing",*Advanced Materials Research,* vol. 457-458, pp. 628-634.

Zhang, J., Yang, Y. and Feng, S., (2012b), "Method of assembly sequence planning based on simulated evolution algorithm",*Advanced Materials Research*, vol. 490, pp. 1171-1175.

Zhang, R., Chen, D., Wang, Y., Yang, Z. and Wang, X. (2007), "Study on line balancing problem based on improved genetic algorithms", *International Conference on Wireless Communications, Networking and Mobile Computing,* 4340283, pp. 2033-2036.

Zhang, W., Gen, M. and Lin, L. (2008), "A multiobjective genetic algorithm for assembly line balancing problem with worker allocation", *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, pp. 3026-3033.

Zhang, Z., Cheng, W., Song, L. and Yu, Q. (2009), "An ant-based algorithm for balancing assembly lines in a mass customization environment",*2009 International Workshop on Intelligent Systems and Applications*, pp. 1-4.

Zhang, Z. and Huang, C., (2012), "A discrete particle swarm optimization approach to optimize the assembly sequence of mechanical product", *Advanced Materials Research,* vol. 490-495, pp. 203-207.

Zhang, Z., Cheng, W., Tang, L. and Zhong, B. (2008), "Ant algorithm with summation rules for assembly line balancing problem", *International Conference on Management Science and Engineering, ICMSE'07*, 4421875, pp. 369-374.

Zhao, L., Li, Y. and Yu, J. (2012), "Assembly sequence unconventionality planning method based on genetic algorithms and gray theory for complex products", *Proceedings - 2012 International Conference on Intelligent Systems Design and Engineering Applications,* pp. 167-171.

Zhao, Z. Y. and De Souza, R. (2000), "Genetic production line-balancing for the hard disk drive industry", *International Journal of Advanced Manufacturing Technology,* vol. 16, no. 4, pp. 297-302.

Zheng, Q., Li, M., Li, Y. and Tang, Q. (2012a), "Station ant colony optimization for the type 2 assembly line balancing problem", *The International Journal of Advanced Manufacturing Technology,* Article in Press, pp. 1-12.

Zheng, Q. X., Li, Y. X., Li, M. and Tang, Q. H. (2012b), "An Improved Ant Colony Optimization for Large-Scale Simple Assembly Line Balancing Problem of Type-1", *Applied Mechanics and Materials,* vol. 159, pp. 51-55.

Zhou, W., Yan, J., Li, Y., Xia, C. and Zheng, J. (2012), "Imperialist competitive algorithm for assembly sequence planning", *The International Journal of Advanced Manufacturing Technology,* Article in Press, pp. 1-10.

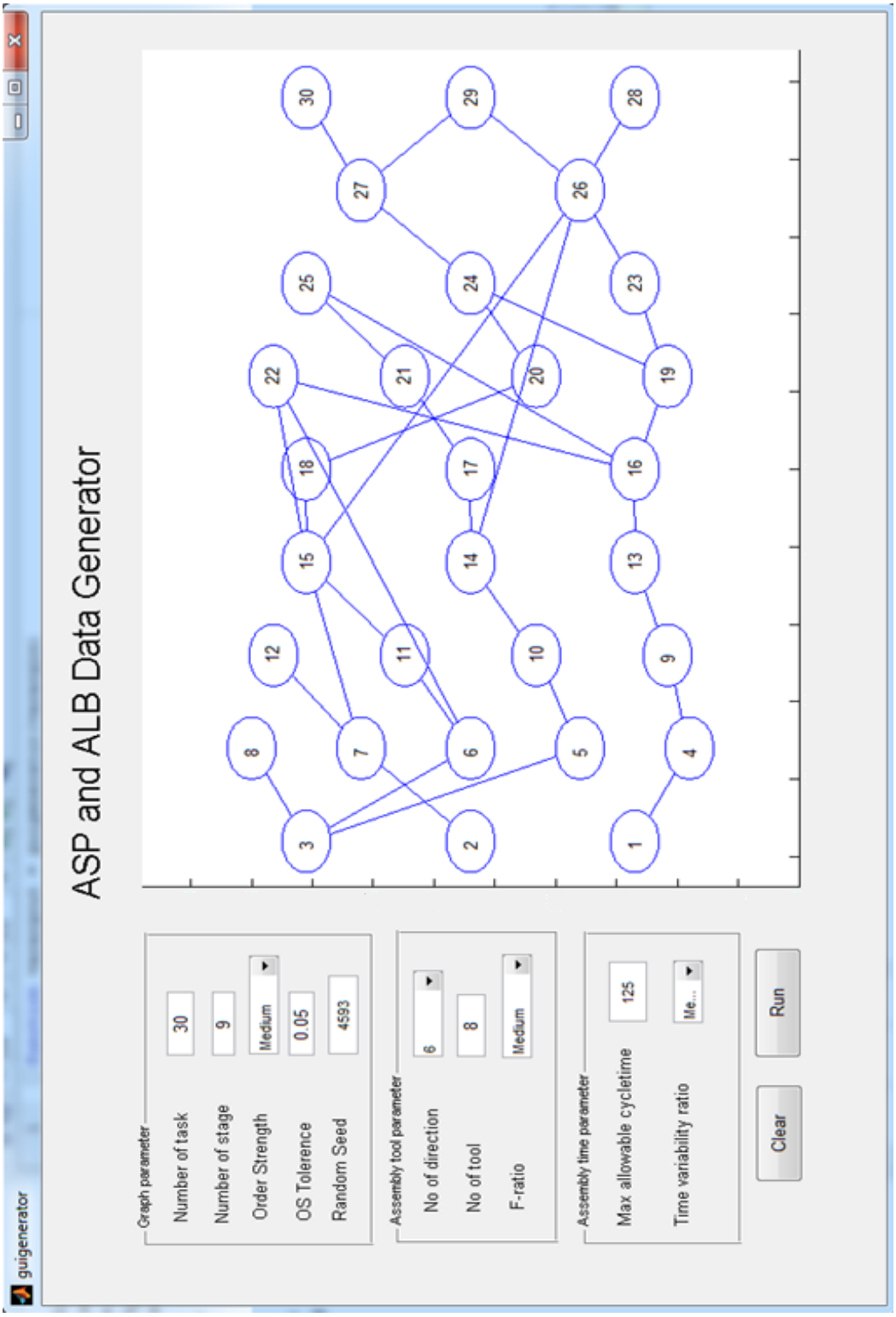
Zhou, W., Zheng, J., Yan, J. and Wang, J. (2011), "A novel hybrid algorithm for assembly sequence planning combining bacterial chemotaxis with genetic algorithm", *International Journal of Advanced Manufacturing Technology,* vol. 52, no. 5-8, pp. 715-724.

Zhu, H. B., Yu, F., Xi, Y., Wang, L. and Zhang, J. (2012a), "Study on Stochastic Assembly Line Balancing Based on Improved Particle Swarm Optimization Algorithm", *Applied Mechanics and Materials,* vol. 130, pp. 3870-3874.

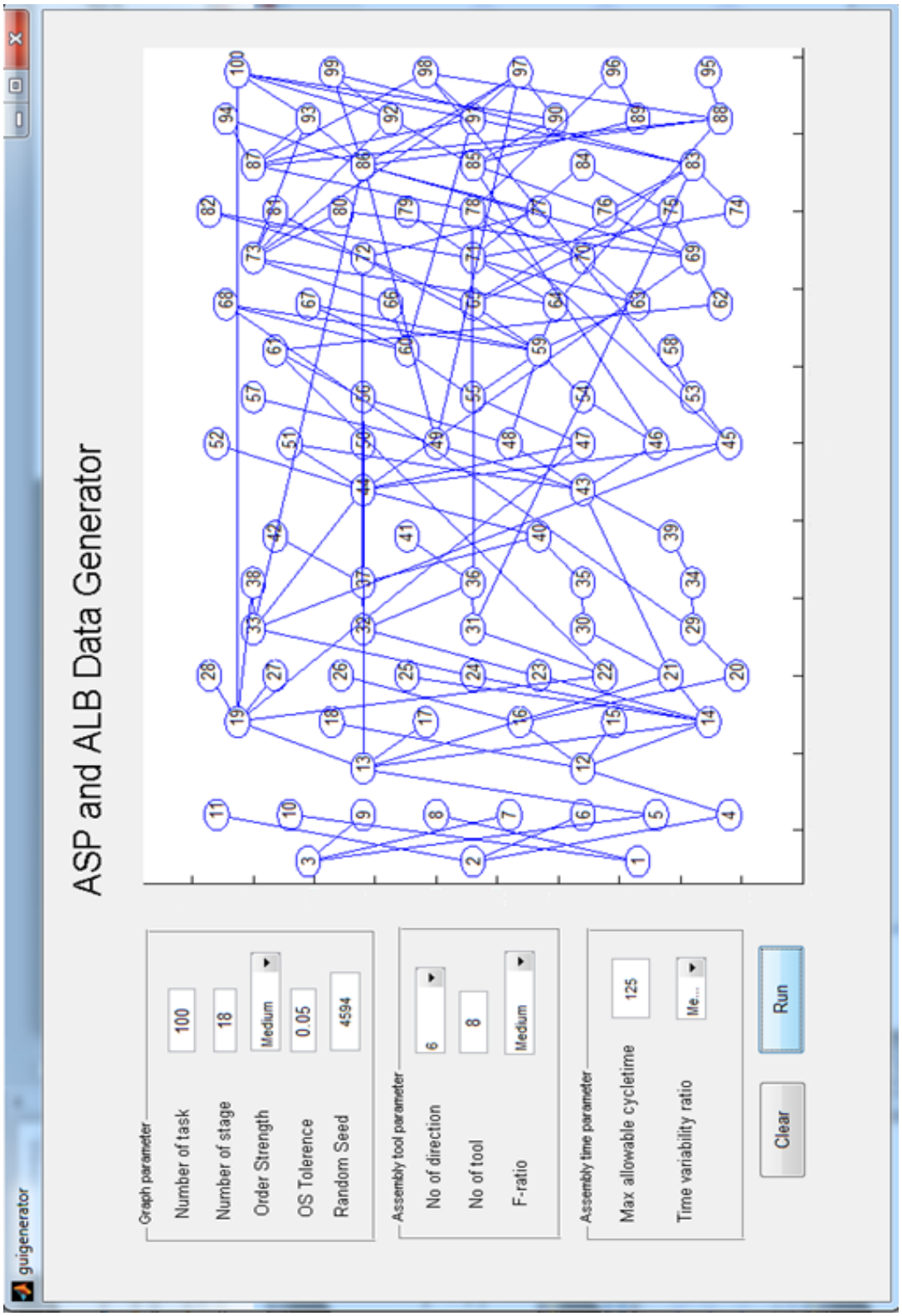
Zhu, X., Hu, S. J., Koren, Y. and Huang, N. (2012b), "A complexity model for sequence planning in mixed-model assembly lines", *Journal of Manufacturing Systems,* vol. 31, no. 2, pp. 121-130.

APPENDIX A: SCREENSHOT OF TUNEABLE TEST PROBLEM GENERATOR

Screenshot of tuneable Test Problem Generator of 30 tasks



Screenshot of tuneable Test Problem Generator of 100 tasks



APPENDIX B: COMPLETE OPTIMISATION RESULTS OF INTEGRATED SINGLE-MODEL ASP AND ALB

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
1	15	0.6	8	0.2	MOGA	9	0.8393	1.3506	1.205	31.8644
					ACO	4	0.9216	1.663	1.3658	31.9873
					HGA	20	0.7059	0.9118	1.1147	31.2991
					NSGA-II	15	0.625	0.8304	1.3034	30.9457
					MOPSO	2	0.9677	1.6477	1.0086	32.5558
					DPSO	1	0.9841	1.9653	1.0051	32.3555
					MODPSO	74	0.1294	0.1875	0.987	31.7432
2	20	0.6	8	0.2	MOGA	3	0.9444	2.439	1.4278	45.5519
					ACO	1	0.9808	2.8606	1.5183	45.1442
					HGA	3	0.9524	2.1352	1.0237	45.241
					NSGA-II	1	0.9737	2.3553	1.9386	45.1442
					MOPSO	0	1	3.1384	1.3064	45.0888
					DPSO	0	1	2.9255	1.1317	44.8596
					MODPSO	63	0.0308	0.0308	1.4427	45.5792
3	40	0.6	8	0.2	MOGA	16	0.8095	2.1461	1.2737	58.3088
					ACO	16	0.7922	2.1457	1.5898	58.1794
					HGA	9	0.91	2.527	1.43	58.1586
					NSGA-II	28	0.2632	0.4871	3.2055	58.5089
					MOPSO	1	0.9921	3.5155	1.3172	57.7791
					DPSO	5	0.9638	3.1515	1.5185	58.829
					MODPSO	58	0.6184	1.5345	1.4769	59.7685
4	60	0.6	8	0.2	MOGA	44	0.6765	1.4765	1.6784	63.4347
					ACO	21	0.7789	1.8998	1.9037	63.5217
					HGA	11	0.9364	2.2504	1.6068	64.7249
					NSGA-II	30	0.375	0.8599	2.2131	62.0252
					MOPSO	9	0.9489	2.588	1.4389	64.8036
					DPSO	36	0.8269	1.8653	1.4292	66.3349
					MODPSO	104	0.6148	1.4023	1.3426	67.4716
5	80	0.6	8	0.2	MOGA	40	0.5876	1.742	4.7713	77.201
					ACO	13	0.8879	2.1749	2.0567	76.1336
					HGA	14	0.9243	2.4662	1.5624	77.9314
					NSGA-II	26	0.5517	1.5131	5.9725	76.7406
					MOPSO	2	0.9914	3.0589	1.3953	77.9206
					DPSO	26	0.8874	2.3105	1.4834	78.3772
					MODPSO	117	0.5412	1.3173	1.9852	80.7361

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
6	15	0.5	8	0.2	MOGA	16	0.7143	1.0564	1.1687	40.638
					ACO	3	0.9375	1.5717	2.3062	40.0404
					HGA	34	0.5	0.6813	1.1524	33.2247
					NSGA-II	21	0.5	0.7658	1.3443	33.0056
					MOPSO	18	0.7429	1.1284	0.9672	40.5065
					DPSO	5	0.9038	1.3976	2.0056	40.2024
					MODPSO	64	0.0986	0.1103	1.8416	39.993
7	15	0.4	8	0.2	MOGA	9	0.6538	0.9882	1.5196	41.2385
					ACO	2	0.9	1.3957	1.3898	42.0286
					HGA	10	0.6875	0.8972	0.8798	41.6007
					NSGA-II	11	0.4762	0.5899	1.2831	40.9805
					MOPSO	3	0.9	1.3496	1.3944	41.2385
					DPSO	2	0.9459	1.7628	1.3708	42.3393
					MODPSO	30	0.0625	0.1141	1.355	42.4101
8	15	0.3	8	0.2	MOGA	10	0.7561	1.0088	1.1828	34.8442
					ACO	4	0.8824	1.4398	1.7417	34.5853
					HGA	15	0.6809	0.853	1.1339	34.556
					NSGA-II	22	0.3529	0.5034	1.567	33.9281
					MOPSO	6	0.8966	1.3593	1.0482	34.28
					DPSO	2	0.9565	1.8167	1.537	34.469
					MODPSO	48	0.1864	0.2691	1.3133	34.8442
9	15	0.2	8	0.2	MOGA	18	0.4194	0.5951	0.9449	38.7186
					ACO	5	0.8	1.2977	2.5045	38.8604
					HGA	25	0.0385	0.0385	2.0952	38.3086
					NSGA-II	19	0.1739	0.1739	2.2721	38.3086
					MOPSO	18	0.3333	0.4074	2.2289	38.5428
					DPSO	3	0.8696	1.1251	1.3611	38.5851
					MODPSO	27	0	0	2.0342	38.3086
10	15	0.6	6	0.2	MOGA	17	0.7119	0.9676	1.0454	31.3031
					ACO	4	0.9245	1.7624	0.9854	31.7713
					HGA	25	0.6324	0.9006	1.2983	37.6464
					NSGA-II	12	0.7143	0.9843	1.5383	30.8065
					MOPSO	6	0.9077	1.4758	1.895	37.9565
					DPSO	4	0.942	1.75	1.0305	38.1274
					MODPSO	50	0.2754	0.3304	1.2144	37.8186
11	15	0.6	4	0.2	MOGA	3	0.9318	2.049	1.6313	32.9716
					ACO	2	0.9545	2.7088	1.4766	32.2665
					HGA	10	0.8077	1.3428	0.8511	33.884
					NSGA-II	7	0.7407	1.2962	1.8183	31.473
					MOPSO	0	1	2.4329	1.0389	33.1682
					DPSO	0	1	3.1728	0.9324	32.0644
					MODPSO	45	0.0816	0.1235	0.7836	32.8044

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
12	15	0.6	3	0.2	MOGA	3	0.9231	1.7957	2.4705	36.6503
					ACO	0	1	2.5366	2.6473	36.3489
					HGA	4	0.9167	1.5554	1.3606	36.9918
					NSGA-II	11	0.4211	0.8022	4.4046	36.1648
					MOPSO	0	1	2.3481	1.2442	36.6652
					DPSO	0	1	2.8293	2.1641	36.7613
					MODPSO	42	0.1429	0.2192	1.1082	36.8255
13	15	0.6	2	0.2	MOGA	11	0.7556	1.1854	1.376	33.6208
					ACO	11	0.6452	1.1913	1.2843	33.4568
					HGA	18	0.5385	0.762	1.2413	33.3251
					NSGA-II	14	0.6	0.9565	2.1024	33.402
					MOPSO	6	0.8909	1.6144	1.0317	33.8136
					DPSO	0	1	2.0216	1.0989	33.789
					MODPSO	40	0.2157	0.2873	0.8782	33.0483
14	15	0.6	8	0.3	MOGA	6	0.6842	0.8549	1.3105	37.8286
					ACO	3	0.8571	1.7677	0.9209	36.9188
					HGA	15	0.2857	0.2857	0.9428	37.6298
					NSGA-II	12	0.3684	0.5866	1.7472	37.4566
					MOPSO	1	0.9655	1.8214	0.8818	37.8286
					DPSO	0	1	2.1749	1.3327	37.3153
					MODPSO	17	0.1905	0.1905	0.992	37.6298
15	15	0.6	8	0.4	MOGA	13	0.74	1.1694	1.1324	51.303
					ACO	2	0.9556	1.7985	1.3783	51.1762
					HGA	18	0.6842	1.0517	1.1566	51.303
					NSGA-II	13	0.7174	0.9963	1.7127	51.303
					MOPSO	7	0.8679	1.2695	0.9478	51.1762
					DPSO	5	0.9091	1.5591	1.0406	51.4107
					MODPSO	60	0.1892	0.2929	0.9425	51.303
16	15	0.6	8	0.6	MOGA	11	0.8136	1.4376	1.4201	47.7285
					ACO	0	1	2.4569	2.2421	47.2982
					HGA	11	0.8254	1.387	1.3004	47.8479
					NSGA-II	10	0.6774	1.2909	1.8822	47.5513
					MOPSO	1	0.9857	2.2844	1.831	47.8696
					DPSO	1	0.9841	2.6918	1.2747	47.5432
					MODPSO	60	0.1892	0.2529	0.8881	47.8448
17	15	0.6	8	0.8	MOGA	9	0.8393	1.3506	1.205	31.8644
					ACO	4	0.9216	1.663	1.3658	31.9873
					HGA	20	0.7059	0.9118	1.1147	31.2991
					NSGA-II	15	0.625	0.8304	1.3034	30.9457
					MOPSO	2	0.9677	1.6477	1.0086	32.5558
					DPSO	1	0.9841	1.9653	1.0051	32.3555
					MODPSO	74	0.1294	0.1875	0.987	31.7432

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
18	40	0.4	8	0.4	MOGA	14	0.9103	2.4789	1.5861	54.0741
					ACO	15	0.881	2.1179	1.3418	53.7692
					HGA	6	0.9718	2.7042	1.2142	53.9493
					NSGA-II	26	0.675	1.555	2.3567	53.754
					MOPSO	7	0.9668	3.0217	1.2171	53.0851
					DPSO	5	0.9755	2.8198	1.3368	54.5667
					MODPSO	196	0.3241	0.6636	1.3787	56.0171
19	15	0.4	4	0.4	MOGA	8	0.8431	1.2089	1.5476	37.6153
					ACO	2	0.9574	1.6998	2.1828	37.7956
					HGA	17	0.6731	0.874	1.3215	37.5678
					NSGA-II	23	0.439	0.6056	2.2323	37.2949
					MOPSO	10	0.8305	1.3036	1.0012	37.6153
					DPSO	3	0.9464	1.6838	1.9206	37.6153
					MODPSO	50	0.1525	0.2344	1.2137	37.7796
20	20	0.4	4	0.4	MOGA	6	0.9333	2.0283	1.5034	40.9673
					ACO	4	0.9403	2.0752	2.141	40.7289
					HGA	7	0.9327	2.0961	1.3901	41.0658
					NSGA-II	11	0.7963	1.4852	1.3818	35.0231
					MOPSO	6	0.9459	2.2467	1.0564	40.8827
					DPSO	3	0.9643	2.3417	1.3526	40.4332
					MODPSO	101	0.0648	0.0885	0.9132	41.1684
21	60	0.4	4	0.4	MOGA	42	0.7705	1.8281	1.7343	63.9951
					ACO	31	0.8208	1.9911	1.5688	64.3157
					HGA	28	0.8857	2.2713	1.7984	63.7421
					NSGA-II	48	0.4725	1.1154	2.4961	63.6739
					MOPSO	7	0.9747	2.8733	1.3271	63.5374
					DPSO	18	0.9357	2.555	1.5142	64.4908
					MODPSO	143	0.5217	1.2151	1.5512	66.8536
22	80	0.4	4	0.4	MOGA	87	0.6506	1.343	1.4958	70.7982
					ACO	61	0.6554	1.3374	2.0642	69.1213
					HGA	49	0.8676	2.0602	1.3692	70.5751
					NSGA-II	11	0.8608	2.3697	2.524	68.8585
					MOPSO	13	0.9637	2.6216	1.2389	71.1042
					DPSO	85	0.7671	1.6628	1.5038	71.1784
					MODPSO	182	0.5956	1.4181	1.493	73.6017
23	40	0.6	4	0.4	MOGA	40	0.7386	1.4114	1.4601	53.7214
					ACO	35	0.7667	1.4999	1.3199	54.2558
					HGA	15	0.9296	2.0287	1.1767	54.0719
					NSGA-II	23	0.7416	1.5464	1.7089	52.0575
					MOPSO	5	0.9782	2.5262	1.0996	53.7675
					DPSO	16	0.9319	2.14	1.2813	53.4767
					MODPSO	177	0.3833	0.7205	1.1945	55.6528

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
24	40	0.5	4	0.4	MOGA	40	0.7386	1.4411	1.3403	53.3354
					ACO	26	0.7969	1.6779	1.249	52.7346
					HGA	18	0.9122	2.0772	1.1696	52.6538
					NSGA-II	23	0.7195	1.6987	2.0723	54.3429
					MOPSO	1	0.9959	2.7617	1.2454	53.7448
					DPSO	11	0.9505	2.2776	1.4327	55.3897
					MODPSO	172	0.3676	0.6828	1.2817	55.2804
25	40	0.3	4	0.4	MOGA	8	0.9527	2.1694	1.5847	53.9902
					ACO	35	0.7107	1.4239	1.4046	53.8765
					HGA	13	0.9414	2.4509	1.2756	54.8341
					NSGA-II	28	0.6957	1.5639	1.913	54.3234
					MOPSO	3	0.9882	3.0331	1.1395	53.8065
					DPSO	4	0.9825	2.7555	1.3321	54.0993
					MODPSO	243	0.211	0.4033	1.4785	56.8437
26	40	0.2	4	0.4	MOGA	18	0.8971	2.4938	1.3965	54.3967
					ACO	15	0.8966	2.1325	1.6584	54.5086
					HGA	9	0.9609	2.7023	1.1858	54.4257
					NSGA-II	25	0.7222	1.3579	1.8767	52.5134
					MOPSO	4	0.9823	3.1216	1.1011	53.4459
					DPSO	6	0.9745	3.0725	1.1964	54.3394
					MODPSO	256	0.1634	0.2709	1.229	55.2653
27	40	0.4	8	0.4	MOGA	21	0.8346	1.956	1.7165	55.2087
					ACO	29	0.7698	1.7133	1.3796	54.1596
					HGA	11	0.9444	2.4682	1.4432	54.1226
					NSGA-II	34	0.5342	1.2222	2.2294	54.9862
					MOPSO	7	0.9646	2.8216	1.1694	54.445
					DPSO	5	0.9738	2.6434	1.3598	54.3415
					MODPSO	123	0.4384	0.9888	1.386	56.088
28	40	0.4	6	0.4	MOGA	29	0.8221	1.8659	1.3524	48.6748
					ACO	31	0.7652	1.6989	1.5974	48.4537
					HGA	11	0.957	2.5945	1.1199	49.5491
					NSGA-II	26	0.6623	1.4491	2.0114	48.4106
					MOPSO	5	0.9796	2.8655	1.3061	48.9429
					DPSO	5	0.9791	2.7222	1.2398	49.2319
					MODPSO	155	0.4106	0.8761	1.279	50.3311
29	40	0.4	3	0.4	MOGA	24	0.85	1.9988	1.553	49.685
					ACO	12	0.9255	2.1254	1.2427	48.7039
					HGA	8	0.9636	2.4581	1.2854	50.209
					NSGA-II	31	0.6961	1.451	1.6457	49.5681
					MOPSO	7	0.9679	2.7638	1.194	48.9853
					DPSO	10	0.955	2.5284	1.3382	50.0456
					MODPSO	194	0.2868	0.5868	1.3869	51.0946

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
30	40	0.4	2	0.4	MOGA	12	0.9273	2.2593	1.3978	47.1965
					ACO	25	0.8062	1.6891	1.5708	45.7039
					HGA	9	0.9561	2.6284	1.3598	47.0084
					NSGA-II	17	0.7821	1.7047	1.819	44.6872
					MOPSO	6	0.9735	3.0683	1.2536	46.9768
					DPSO	8	0.9683	2.9815	1.1787	46.9171
					MODPSO	198	0.2826	0.5659	1.2032	48.726
31	40	0.4	4	0.2	MOGA	26	0.8385	1.6461	1.857	55.1886
					ACO	45	0.7115	1.3985	1.3498	53.4767
					HGA	13	0.9393	2.3758	1.2341	55.0159
					NSGA-II	31	0.6556	1.6401	1.7858	53.7875
					MOPSO	6	0.973	2.7145	1.2116	53.4435
					DPSO	3	0.9876	2.5732	1.1435	52.988
					MODPSO	180	0.404	0.7605	1.2214	56.3423
32	40	0.4	4	0.3	MOGA	10	0.9301	2.3297	2.0146	53.4299
					ACO	16	0.8889	2.2793	1.3974	52.72
					HGA	9	0.9524	2.593	1.3354	53.4926
					NSGA-II	40	0.3939	0.8567	2.4645	52.5337
					MOPSO	4	0.9815	2.9975	1.2896	54.8446
					DPSO	5	0.9773	2.9704	1.2622	54.355
					MODPSO	123	0.5341	1.2277	1.3132	55.1762
33	40	0.4	4	0.6	MOGA	19	0.8681	2.0101	1.6981	53.6721
					ACO	14	0.9021	2.111	1.5515	52.3621
					HGA	13	0.933	2.4548	1.2932	54.312
					NSGA-II	25	0.6988	1.4765	1.8696	52.8108
					MOPSO	7	0.9714	3.0575	1.337	54.9558
					DPSO	4	0.9831	2.7728	1.0868	53.2424
					MODPSO	162	0.3468	0.7617	1.5208	55.0748
34	40	0.4	4	0.8	MOGA	33	0.807	1.9818	1.446	54.8269
					ACO	39	0.7622	1.7333	1.4283	54.6113
					HGA	10	0.9569	2.356	1.3319	53.8867
					NSGA-II	39	0.5568	1.1792	1.9603	53.2672
					MOPSO	9	0.9615	2.8303	1.2316	54.6899
					DPSO	7	0.9715	2.7965	1.207	53.8682
					MODPSO	144	0.5184	1.1797	1.1127	55.6936
35	80	0.2	2	0.8	MOGA	33	0.8012	2.1613	1.9462	77.3322
					ACO	41	0.6772	1.7737	2.3191	76.4089
					HGA	18	0.9244	2.7463	1.5664	76.4399
					NSGA-II	39	0.4868	1.6774	2.9249	74.8999
					MOPSO	10	0.9635	3.2856	1.5333	76.181
					DPSO	31	0.8848	2.609	1.8031	75.9732
					MODPSO	143	0.5903	1.5209	1.5347	79.355

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
36	15	0.2	2	0.8	MOGA	15	0.2857	0.4709	1.4851	37.9987
					ACO	10	0.5238	0.8408	2.0705	38.3163
					HGA	21	0.1923	0.2242	1.4781	38.1987
					NSGA-II	13	0.2353	0.3668	1.443	37.7876
					MOPSO	14	0.2632	0.4296	1.2253	37.9987
					DPSO	9	0.5263	0.8103	1.0745	38.1301
					MODPSO	23	0.08	0.1366	0.6738	38.0119
37	20	0.2	2	0.8	MOGA	8	0.8571	1.5927	0.8464	32.8465
					ACO	0	1	2.0884	1.3505	33.074
					HGA	11	0.7885	1.3653	1	32.8034
					NSGA-II	20	0.4286	0.654	1.4496	32.4971
					MOPSO	4	0.9375	1.6994	0.9715	33.074
					DPSO	5	0.9091	1.8169	1.4372	33.1789
					MODPSO	40	0.3103	0.4247	1.1264	32.6582
38	40	0.2	2	0.8	MOGA	10	0.9286	2.5731	1.7664	57.9728
					ACO	9	0.9217	2.151	1.963	50.9298
					HGA	16	0.9106	2.5991	1.5298	58.0518
					NSGA-II	53	0.4752	0.9906	1.4947	52.5908
					MOPSO	7	0.9626	3.1257	1.3334	57.4683
					DPSO	4	0.9781	2.8587	1.5826	57.9673
					MODPSO	139	0.4009	0.7895	1.3824	59.3718
39	60	0.2	2	0.8	MOGA	21	0.8091	2.1781	2.0076	58.9908
					ACO	15	0.8558	2.3812	1.953	57.9541
					HGA	20	0.8726	2.2923	1.6545	57.1094
					NSGA-II	18	0.6087	1.5812	2.7851	56.6142
					MOPSO	7	0.9641	3.0485	1.5183	57.337
					DPSO	15	0.9231	2.6471	1.3575	58.2652
					MODPSO	101	0.5826	1.5188	1.5878	60.6911
40	80	0.6	2	0.8	MOGA	57	0.6851	1.5658	1.7144	70.5479
					ACO	64	0.5556	1.2475	1.7865	70.8649
					HGA	28	0.8871	2.0777	1.6059	69.99
					NSGA-II	13	0.7969	1.7609	2.2271	67.3636
					MOPSO	18	0.9277	2.4269	1.5151	69.9913
					DPSO	61	0.7829	1.6865	1.4383	70.5073
					MODPSO	176	0.4568	0.9854	1.6428	72.1917
41	80	0.5	2	0.8	MOGA	27	0.8015	1.9844	1.9384	66.2236
					ACO	37	0.791	1.908	1.9421	66.83
					HGA	17	0.9274	2.3605	1.52	66.2875
					NSGA-II	24	0.6364	1.4994	2.8245	64.7821
					MOPSO	11	0.9556	2.9266	1.4146	65.6162
					DPSO	26	0.885	2.0759	1.3299	65.7862
					MODPSO	147	0.4842	1.0731	1.4993	69.1016

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
42	80	0.4	2	0.8	MOGA	18	0.875	2.3642	1.8808	58.9744
					ACO	34	0.7213	1.6564	1.6835	55.7823
					HGA	28	0.8931	2.451	1.5194	60.5438
					NSGA-II	20	0.7222	2.0927	3.2246	58.8101
					MOPSO	9	0.9654	2.9841	1.4934	59.9632
					DPSO	35	0.8638	2.3167	1.2642	58.7422
					MODPSO	137	0.5018	1.2609	1.5604	61.5433
43	80	0.3	2	0.8	MOGA	28	0.8436	2.3577	1.5605	66.1466
					ACO	40	0.7484	1.8763	1.7801	66.1274
					HGA	21	0.9211	2.5509	1.7278	67.6063
					NSGA-II	29	0.6742	1.7512	3.0234	66.9709
					MOPSO	9	0.971	3.0234	1.4115	67.6984
					DPSO	48	0.8222	2.2148	1.6482	66.5509
					MODPSO	171	0.4063	0.947	1.7489	71.3896
44	80	0.2	8	0.8	MOGA	38	0.7099	2.4201	2.5364	77.3515
					ACO	13	0.8879	2.8836	2.0446	75.9013
					HGA	15	0.9261	3.0986	1.5162	76.6297
					NSGA-II	47	0.338	0.8921	2.9603	75.9163
					MOPSO	6	0.9766	3.5263	1.4183	77.1572
					DPSO	18	0.9379	3.2396	1.7651	79.2179
					MODPSO	85	0.684	1.9675	1.9916	80.6747
45	80	0.2	6	0.8	MOGA	24	0.8769	2.5157	1.7039	72.1097
					ACO	73	0.4859	1.1801	2.3302	73.1415
					HGA	25	0.906	2.7115	1.6459	72.564
					NSGA-II	45	0.4444	1.1525	3.3866	73.2854
					MOPSO	15	0.951	2.9482	1.5142	73.0103
					DPSO	46	0.8726	2.4714	1.4534	73.8751
					MODPSO	123	0.6854	1.8551	1.56	75.103
46	80	0.2	4	0.8	MOGA	22	0.8571	2.5773	1.9121	76.5617
					ACO	34	0.6822	1.8255	2.2181	74.4104
					HGA	20	0.9156	2.6566	1.7609	75.2983
					NSGA-II	43	0.4267	1.0253	2.7963	74.7239
					MOPSO	9	0.9656	2.9892	1.7972	75.5546
					DPSO	38	0.8841	2.648	1.6713	77.4392
					MODPSO	144	0.5034	1.3304	1.8496	76.8449
47	80	0.2	3	0.8	MOGA	30	0.8246	2.1902	2.131	69.8498
					ACO	20	0.8387	2.0828	1.8019	66.9948
					HGA	23	0.9102	2.6714	1.7583	69.8999
					NSGA-II	47	0.2985	1.0786	3.2647	71.0533
					MOPSO	7	0.9739	3.1957	1.6635	69.3332
					DPSO	30	0.8824	2.4533	1.5359	68.5087
					MODPSO	116	0.6246	1.6752	1.9537	72.0142

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
48	80	0.2	2	0.2	MOGA	24	0.8588	2.3307	2.1236	68.7445
					ACO	63	0.642	1.6521	1.9866	69.6054
					HGA	17	0.9395	2.7722	1.5928	69.3235
					NSGA-II	32	0.6098	1.8048	2.9211	71.2461
					MOPSO	15	0.9519	3.0469	1.5399	68.7935
					DPSO	22	0.9231	2.7719	1.5323	69.0669
					MODPSO	143	0.4781	1.269	1.5938	70.8176
49	80	0.2	2	0.3	MOGA	16	0.8779	2.7858	2.0651	68.0726
					ACO	6	0.9641	2.9501	1.7691	67.1528
					HGA	7	0.9657	3.5136	1.6471	69.0634
					NSGA-II	29	0.6463	1.7287	2.3512	71.2643
					MOPSO	6	0.9795	3.8815	1.5425	68.3256
					DPSO	13	0.95	3.002	1.7944	68.9116
					MODPSO	223	0.2203	0.5712	1.9388	73.3672
50	80	0.2	2	0.4	MOGA	9	0.9571	3.0417	1.8731	69.8832
					ACO	15	0.9026	2.5183	1.6617	66.8588
					HGA	15	0.94	3.044	1.4688	67.4671
					NSGA-II	16	0.7895	2.4666	2.7625	69.3856
					MOPSO	11	0.9596	3.3687	1.6103	69.1744
					DPSO	17	0.9404	3.0772	1.6409	68.3236
					MODPSO	211	0.2038	0.5683	1.8414	73.0428
51	80	0.2	2	0.6	MOGA	24	0.8613	2.4262	1.8821	68.15
					ACO	14	0.9091	2.7658	1.7689	66.1974
					HGA	7	0.975	3.3882	1.4852	70.229
					NSGA-II	26	0.6977	2.0647	2.8891	69.9303
					MOPSO	5	0.9829	3.6828	1.7177	70.2157
					DPSO	17	0.9356	2.7497	1.7017	69.6885
					MODPSO	240	0.2053	0.488	1.7855	71.486

APPENDIX C: COMPLETE OPTIMISATION RESULTS OF INTEGRATED MIXED-MODEL ASP AND ALB

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
1	15	0.6	8	0.2	MOGA	10	0.697	0.5916	1.0768	18.3606
					ACO	0	1	1.1502	1.6022	18.8444
					HGA	18	0.5714	0.4271	0.6619	17.9633
					NSGA-II	26	0.1034	0.0488	1.0713	17.3338
					MOPSO	11	0.7381	0.6541	0.965	17.9253
					DPSO	6	0.8235	0.6691	0.927	17.2115
					MODPSO	18	0.6087	0.567	0.8806	18.421
2	20	0.6	8	0.2	MOGA	3	0.9531	1.0425	0.7454	14.1188
					ACO	2	0.96	1.5017	0.8807	15.6461
					HGA	11	0.8136	0.787	0.96	15.5579
					NSGA-II	38	0.1364	0.1094	0.9626	15.4029
					MOPSO	4	0.956	1.2329	0.8436	16.4392
					DPSO	5	0.918	1.1618	1.0186	16.4692
					MODPSO	47	0.53	0.5133	0.7867	18.6739
3	40	0.6	8	0.2	MOGA	13	0.8194	1.5805	0.8989	18.6011
					ACO	0	1	2.7723	1.5243	13.8149
					HGA	10	0.8246	1.4189	1.5237	16.9571
					NSGA-II	33	0.0294	0.0294	1.1091	13.6545
					MOPSO	3	0.9589	2.3001	1.013	18.1258
					DPSO	2	0.9798	2.4725	0.8615	17.0657
					MODPSO	48	0.0588	0.0554	1.531	18.5587
4	60	0.6	8	0.2	MOGA	2	0.9747	4.0025	0.9303	15.4696
					ACO	0	1	4.6506	1.1451	13.9293
					HGA	0	1	3.6967	1.1781	16.552
					NSGA-II	47	0	0	1.0114	15.1959
					MOPSO	1	0.9907	4.4718	0.9036	17.0017
					DPSO	0	1	4.6788	0.7729	15.8072
					MODPSO	49	0	0	1.9834	18.4096
5	80	0.6	8	0.2	MOGA	5	0.9275	6.1246	3.1172	43.7845
					ACO	3	0.9583	6.3362	2.353	41.5159
					HGA	2	0.9726	7.5129	3.0304	42.9069
					NSGA-II	26	0	0	5.6045	37.9987
					MOPSO	3	0.9778	8.4623	2.5643	49.6
					DPSO	3	0.9778	8.4623	2.5643	49.6
					MODPSO	36	0.234	1.0442	3.8141	59.4803

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
6	15	0.5	8	0.2	MOGA	8	0.8545	0.6973	1.6039	19.8105
					ACO	0	1	1.4073	1.8745	17.4827
					HGA	16	0.7778	0.584	1.082	18.6452
					NSGA-II	20	0.4286	0.2978	1.0037	13.2223
					MOPSO	12	0.8356	0.9374	0.8573	19.0266
					DPSO	5	0.9398	1.0533	0.9839	21.8822
					MODPSO	57	0.4	0.2422	0.7909	18.4535
7	15	0.4	8	0.2	MOGA	12	0.6842	0.5792	1.0826	16.8529
					ACO	1	0.973	1.3256	1.2758	14.8295
					HGA	16	0.68	0.562	1.0474	14.2689
					NSGA-II	15	0.5946	0.4747	0.9251	14.2095
					MOPSO	11	0.8358	1.0176	0.8199	19.0111
					DPSO	6	0.8846	1.0857	1.3486	18.0173
					MODPSO	41	0.2807	0.1869	0.8348	17.4736
8	15	0.3	8	0.2	MOGA	9	0.8831	0.7771	0.9296	20.4773
					ACO	1	0.9811	1.2792	0.8381	17.781
					HGA	41	0.5684	0.4341	0.8784	22.0404
					NSGA-II	17	0.6047	0.4508	1.2362	18.1656
					MOPSO	24	0.7624	0.6103	0.8632	22.5326
					DPSO	9	0.9032	0.9178	1.1024	23.4664
					MODPSO	71	0.3772	0.3568	0.9225	21.3959
9	15	0.2	8	0.2	MOGA	7	0.8814	0.8397	0.6895	16.3336
					ACO	2	0.9556	1.6646	0.8102	17.8686
					HGA	21	0.5882	0.5557	0.9622	17.0313
					NSGA-II	6	0.8378	0.9535	1.4187	17.1659
					MOPSO	6	0.9048	1.1233	0.9622	18.0819
					DPSO	2	0.9565	1.1168	0.8719	17.4444
					MODPSO	45	0.3182	0.2439	0.7305	14.6099
10	15	0.6	6	0.2	MOGA	9	0.7805	0.6027	0.9875	14.2928
					ACO	1	0.9697	1.1137	1.5427	14.2539
					HGA	12	0.7143	0.6117	0.8896	13.9629
					NSGA-II	19	0.4242	0.311	1.1523	13.9863
					MOPSO	3	0.9444	0.9416	0.7277	15.3963
					DPSO	3	0.9362	1.072	0.9446	14.7031
					MODPSO	40	0.1304	0.0848	0.8381	15.3931
11	15	0.6	4	0.2	MOGA	16	0.6863	0.4933	0.9198	16.4607
					ACO	0	1	1.1417	0.6771	13.4334
					HGA	9	0.8831	0.7188	0.8356	16.043
					NSGA-II	19	0.5	0.4929	1.1271	16.7179
					MOPSO	13	0.8	0.7968	0.8733	17.0679
					DPSO	5	0.9333	1.1805	0.7851	17.2211
					MODPSO	34	0.4848	0.3915	0.8253	17.6011

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
12	15	0.6	3	0.2	MOGA	10	0.7917	0.5087	0.7989	14.9982
					ACO	36	0.425	0.2736	0.4705	15.6623
					HGA	21	0.6866	0.4347	0.9122	15.7499
					NSGA-II	21	0.3636	0.2544	0.9076	13.3209
					MOPSO	14	0.7846	0.4878	0.8698	16.1107
					DPSO	11	0.8308	0.5123	0.8668	16.0573
					MODPSO	40	0.3939	0.2437	0.6387	15.2917
13	15	0.6	2	0.2	MOGA	6	0.7857	0.5789	1.3236	13.1276
					ACO	1	0.963	0.8037	1.1538	12.6973
					HGA	13	0.6061	0.4386	1.1905	14.2322
					NSGA-II	15	0.4643	0.455	1.0133	14.1539
					MOPSO	8	0.7949	0.5661	1.0938	14.9109
					DPSO	4	0.8889	0.6234	0.6127	14.9406
					MODPSO	30	0.2683	0.1128	1.1655	13.3334
14	15	0.6	8	0.3	MOGA	14	0.5882	0.3516	0.8034	13.6938
					ACO	3	0.8889	0.9695	0.7298	13.4247
					HGA	18	0.5385	0.3285	0.6158	13.6937
					NSGA-II	14	0.5	0.3683	1.2378	13.7477
					MOPSO	25	0.4048	0.2186	0.8521	15.6226
					DPSO	16	0.5556	0.3361	0.7793	14.2244
					MODPSO	32	0.36	0.1911	0.7466	15.6276
15	15	0.6	8	0.4	MOGA	11	0.6857	0.5918	0.9063	14.3605
					ACO	2	0.9048	0.874	1.4687	11.7426
					HGA	19	0.5	0.4192	0.404	13.9881
					NSGA-II	15	0.4231	0.3158	1.3436	13.0043
					MOPSO	10	0.7778	0.5419	0.4095	14.5527
					DPSO	9	0.8163	0.736	0.6734	14.4952
					MODPSO	30	0.25	0.1328	0.711	13.7316
16	15	0.6	8	0.6	MOGA	8	0.8182	0.6593	0.7268	15.9616
					ACO	2	0.931	0.9482	1.0481	15.3985
					HGA	11	0.8254	0.7886	1.0157	17.1835
					NSGA-II	8	0.7419	0.6105	1.1662	13.6666
					MOPSO	2	0.9701	0.9745	0.9023	16.4417
					DPSO	4	0.9259	1.0332	0.8682	16.7364
					MODPSO	45	0.3077	0.2344	0.9408	16.8457
17	15	0.6	8	0.8	MOGA	14	0.7143	0.474	0.7301	13.2513
					ACO	0	1	1.1758	0.6031	10.2177
					HGA	29	0.5	0.3285	0.7264	13.5347
					NSGA-II	14	0.5	0.2907	0.7726	11.3422
					MOPSO	15	0.7	0.4803	0.9951	13.2111
					DPSO	11	0.807	0.6522	0.678	12.8296
					MODPSO	50	0.0741	0.0349	0.7246	13.8673

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
18	40	0.4	8	0.4	MOGA	10	0.899	2.1711	0.9479	19.5265
					ACO	2	0.9737	2.5197	0.8166	16.8391
					HGA	11	0.89	1.7912	0.9275	18.7966
					NSGA-II	23	0.2813	0.309	1.2089	14.7327
					MOPSO	11	0.9167	2.2219	1.0091	21.6984
					DPSO	3	0.9746	2.1845	1.0692	19.0578
					MODPSO	27	0.784	1.9532	1.0197	21.282
19	15	0.4	4	0.4	MOGA	7	0.8793	0.8245	0.8711	18.8363
					ACO	0	1	1.1054	0.9471	15.4533
					HGA	13	0.7969	0.7097	0.9595	19.2246
					NSGA-II	15	0.6053	0.5697	1.2268	17.5318
					MOPSO	8	0.8919	0.7681	0.8067	20.8733
					DPSO	3	0.9583	0.8707	1.0398	18.9334
					MODPSO	59	0.2716	0.235	0.6967	19.1319
20	20	0.4	4	0.4	MOGA	9	0.8125	0.8242	1.0769	19.4841
					ACO	0	1	1.4778	1.3172	17.2791
					HGA	12	0.8421	0.7541	0.6489	19.0328
					NSGA-II	10	0.7368	0.701	1.3386	18.2124
					MOPSO	3	0.9677	1.1314	0.6647	19.0607
					DPSO	1	0.9846	1.2103	0.8868	18.269
					MODPSO	58	0.383	0.3104	0.8758	20.9093
21	60	0.4	4	0.4	MOGA	3	0.9388	3.0454	1.527	15.3074
					ACO	1	0.9783	3.5356	1.1852	13.2487
					HGA	0	1	3.6186	0.9554	16.7889
					NSGA-II	29	0	0	1.5061	13.0229
					MOPSO	0	1	3.8769	0.7935	15.7795
					DPSO	2	0.9744	3.451	0.9404	16.6345
					MODPSO	32	0.0303	0.105	1.874	18.398
22	80	0.4	4	0.4	MOGA	0	1	3.8727	1.714	16.7242
					ACO	0	1	4.206	1.1865	18.5788
					HGA	3	0.9712	4.258	1.2393	19.6896
					NSGA-II	4	0.8667	0.0001	1.0132	13.1838
					MOPSO	0	1	4.014	1.1609	21.2317
					DPSO	0	1	4.1059	1.0403	20.8324
					MODPSO	6	0.962	4.0225	1.046	22.4549
23	40	0.6	4	0.4	MOGA	1	0.9796	2.5813	1.1387	15.6408
					ACO	0	1	2.986	1.1745	15.1647
					HGA	4	0.9412	2.3205	0.9757	16.9643
					NSGA-II	39	0	0	1.253	14.8374
					MOPSO	5	0.9405	2.953	0.8947	16.4962
					DPSO	2	0.9683	2.6726	1.0492	16.8523
					MODPSO	35	0.025	0.0417	1.8231	17.7337

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
24	40	0.5	4	0.4	MOGA	6	0.9048	2.4881	0.961	15.8622
					ACO	0	1	3.4289	0.9044	15.7229
					HGA	1	0.9828	2.4555	1.3104	15.6518
					NSGA-II	39	0	0	1.0336	13.1656
					MOPSO	2	0.9792	2.8368	1.0464	16.2278
					DPSO	2	0.9767	2.9332	1.0043	16.4342
					MODPSO	37	0	0	1.4864	14.6326
25	40	0.3	4	0.4	MOGA	4	0.9481	2.8835	0.9153	16.2965
					ACO	0	1	3.3995	1.0277	12.9824
					HGA	2	0.9683	2.81	1.2604	15.8323
					NSGA-II	34	0	0	1.3593	15.3186
					MOPSO	5	0.9474	2.9951	0.9677	16.5799
					DPSO	3	0.962	2.641	1.0522	17.2783
					MODPSO	12	0.8824	2.5956	0.8983	18.5687
26	40	0.2	4	0.4	MOGA	0	1	2.4549	1.0045	16.2609
					ACO	0	1	3.2388	0.9937	16.1538
					HGA	3	0.9677	2.2773	1.0622	17.578
					NSGA-II	39	0.0714	0.0743	1.6105	19.0912
					MOPSO	2	0.9825	2.7261	0.9362	17.9267
					DPSO	1	0.9901	2.6546	0.9773	17.7349
					MODPSO	59	0	0	1.4265	22.1047
27	40	0.4	8	0.4	MOGA	0	1	2.5172	1.0203	14.1707
					ACO	2	0.9512	3.0215	1.257	15.3152
					HGA	3	0.9545	2.7692	0.7858	15.8268
					NSGA-II	34	0	0	1.6317	17.0495
					MOPSO	1	0.9878	3.2051	0.9292	17.1425
					DPSO	3	0.9677	3.1323	0.9581	17.7162
					MODPSO	42	0.0455	0.0397	1.8739	19.5834
28	40	0.4	6	0.4	MOGA	8	0.8519	1.5532	1.2292	14.0885
					ACO	0	1	2.8022	1.11	14.1697
					HGA	6	0.9362	1.992	1.0712	17.764
					NSGA-II	35	0.125	0.1514	1.4949	17.0199
					MOPSO	9	0.8889	2.2868	1.0431	17.1865
					DPSO	3	0.9754	2.4189	0.874	17.7952
					MODPSO	63	0.0156	0.009	1.2536	18.9552
29	40	0.4	3	0.4	MOGA	0	1	1.8258	1.4205	14.7874
					ACO	0	1	2.8895	1.0831	12.7219
					HGA	7	0.9091	2.1263	0.9068	16.63
					NSGA-II	36	0	0	1.0585	16.6049
					MOPSO	3	0.9752	2.6477	1.1504	17.6061
					DPSO	1	0.9872	2.6085	1.0138	16.6803
					MODPSO	56	0	0	1.2056	20.5702

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
30	40	0.4	2	0.4	MOGA	1	0.9821	1.8225	1.0738	14.4192
					ACO	0	1	2.3179	0.9921	13.3032
					HGA	9	0.9072	1.5806	1.0811	17.4134
					NSGA-II	31	0.0882	0.0813	1.719	15.7398
					MOPSO	10	0.9107	1.9498	0.8107	16.8315
					DPSO	3	0.9712	2.1865	0.7812	15.547
					MODPSO	70	0.0278	0.0353	1.3239	18.5596
31	40	0.4	4	0.2	MOGA	2	0.9623	3.3399	0.9144	13.6707
					ACO	0	1	3.4839	1.3369	13.944
					HGA	3	0.95	3.156	0.9681	14.1261
					NSGA-II	30	0	0	1.4296	13.7598
					MOPSO	3	0.962	3.361	1.0082	15.2717
					DPSO	0	1	3.4557	1.0409	15.5129
					MODPSO	39	0.0488	0.0539	1.4707	15.9834
32	40	0.4	4	0.3	MOGA	4	0.9467	2.1652	0.8768	17.7072
					ACO	0	1	2.7146	1.1265	13.6837
					HGA	4	0.942	1.9978	1.0226	17.0556
					NSGA-II	38	0.0256	0.0121	1.3339	13.3141
					MOPSO	2	0.9756	2.3313	1.2076	17.5155
					DPSO	4	0.9579	2.2597	1.0647	17.4721
					MODPSO	47	0.0784	0.0839	1.3854	16.4463
33	40	0.4	4	0.6	MOGA	0	1	3.0105	1.0273	13.7032
					ACO	0	1	3.4301	1.4675	15.0813
					HGA	1	0.9844	2.5285	1.2436	17.7675
					NSGA-II	40	0	0	1.1501	14.1454
					MOPSO	2	0.9773	3.3419	1.0842	19.4027
					DPSO	1	0.988	3.5069	1.1796	18.5316
					MODPSO	42	0.0455	0.0865	1.6836	17.2645
34	40	0.4	4	0.8	MOGA	2	0.9688	2.713	0.7265	13.7927
					ACO	3	0.9583	2.8922	0.7463	13.8494
					HGA	3	0.9643	2.9079	0.8141	15.0774
					NSGA-II	29	0.0333	0.0351	1.8121	14.3229
					MOPSO	1	0.9836	2.7428	0.953	15.4511
					DPSO	3	0.9545	3.2316	1.068	14.7397
					MODPSO	32	0.0303	0.0319	1.5837	15.3707
35	80	0.2	2	0.8	MOGA	0	1	2.1585	1.0363	15.3171
					ACO	0	1	2.7418	1.8854	18.342
					HGA	0	1	2.568	1.1264	20.0434
					NSGA-II	9	0.7632	0.0001	1.2368	16.0324
					MOPSO	0	1	2.794	0.8499	17.6503
					DPSO	1	0.9873	2.9067	0.9853	18.1218
					MODPSO	9	0.9338	2.7781	1.0778	21.6836

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
36	15	0.2	2	0.8	MOGA	1	0.973	0.7657	0.8141	12.741
					ACO	0	1	1.2817	0.6271	11.2689
					HGA	5	0.878	0.6931	0.6665	14.5641
					NSGA-II	19	0.4571	0.323	1.3745	13.4413
					MOPSO	1	0.9792	1.1138	1.1685	16.5924
					DPSO	2	0.9333	1.017	0.7115	12.6448
					MODPSO	49	0.3875	0.267	0.7762	17.7296
37	20	0.2	2	0.8	MOGA	1	0.9615	1.3867	0.9516	11.3946
					ACO	0	1	1.8508	0.663	10.8179
					HGA	8	0.6	0.6291	1.1219	11.55
					NSGA-II	10	0.4444	0.3961	1.455	12.0239
					MOPSO	0	1	2.0077	1.1936	11.4202
					DPSO	0	1	2.283	0.9987	11.6074
					MODPSO	13	0.6977	0.7362	0.9091	15.8203
38	40	0.2	2	0.8	MOGA	6	0.9016	1.8385	1.0848	13.6532
					ACO	5	0.8438	1.7837	1.2038	12.6825
					HGA	11	0.8103	1.466	0.6636	12.5284
					NSGA-II	36	0.027	0.0525	1.3985	12.7802
					MOPSO	3	0.9655	2.4254	0.7799	13.4325
					DPSO	4	0.942	2.3629	1.0843	13.557
					MODPSO	44	0.12	0.1842	1.2072	15.9985
39	60	0.2	2	0.8	MOGA	0	1	3.0393	0.9836	10.3723
					ACO	1	0.9565	2.4008	0.9299	8.1133
					HGA	0	1	2.9031	0.6635	11.8288
					NSGA-II	28	0	0	0.8903	10.6921
					MOPSO	0	1	2.8865	0.8147	12.7677
					DPSO	1	0.9767	3.1311	0.7725	11.5436
					MODPSO	33	0.1081	0.3368	1.3958	17.7705
40	80	0.6	2	0.8	MOGA	0	1	5.0654	0.9033	18.7657
					ACO	2	0.963	4.7725	0.7864	12.9279
					HGA	1	0.9851	4.8319	0.8687	19.3387
					NSGA-II	40	0	0	0.8006	13.2957
					MOPSO	3	0.9754	4.7045	1.0807	20.0764
					DPSO	1	0.9894	5.041	0.9712	18.6855
					MODPSO	46	0.0213	0.0123	0.8537	23.4828
41	80	0.5	2	0.8	MOGA	5	0.9451	2.1061	0.9474	20.0264
					ACO	0	1	2.6753	0.9614	17.0608
					HGA	6	0.9459	2.5204	0.9665	20.1406
					NSGA-II	30	0.0323	0.0493	1.8475	21.0621
					MOPSO	10	0.9375	2.6579	1.1123	22.7167
					DPSO	1	0.9894	2.4669	0.9307	19.0937
					MODPSO	47	0.1132	0.1708	1.362	25.4767

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
42	80	0.4	2	0.8	MOGA	1	0.9851	3.0119	0.9629	16.4184
					ACO	6	0.8125	2.3249	2.1567	15.3431
					HGA	5	0.9468	3.1665	0.8713	18.7401
					NSGA-II	36	0	0	1.6622	15.5567
					MOPSO	9	0.9032	3.1892	1.0828	17.88
					DPSO	5	0.9565	3.445	1.0476	18.5761
					MODPSO	42	0.1064	0.1194	1.535	21.4433
43	80	0.3	2	0.8	MOGA	1	0.9841	3.1339	1.1742	15.9324
					ACO	0	1	3.6016	1.797	16.0681
					HGA	2	0.9759	3.3242	1.0737	17.4185
					NSGA-II	30	0	0	1.2464	14.6554
					MOPSO	0	1	3.552	1.1168	18.1068
					DPSO	1	0.9907	3.5661	0.9506	17.195
					MODPSO	35	0.0789	0.1898	1.4687	22.7499
44	80	0.2	8	0.8	MOGA	3	0.9388	2.0557	1.1162	14.5111
					ACO	2	0.9623	2.6911	1.3545	15.6847
					HGA	1	0.9868	2.9253	0.9301	16.2099
					NSGA-II	35	0.2045	0.5062	1.185	16.5852
					MOPSO	0	1	2.947	1.0973	16.7928
					DPSO	0	1	3.0007	1.058	16.3665
					MODPSO	43	0.1887	0.4292	0.9919	18.0481
45	80	0.2	6	0.8	MOGA	0	1	3.448	1.1333	14.27
					ACO	0	1	3.2766	1.3113	13.4292
					HGA	3	0.9412	3.2161	1.2048	15.399
					NSGA-II	23	0	0	1.0336	12.9598
					MOPSO	2	0.9737	3.2324	1.1174	15.2885
					DPSO	2	0.9787	3.5149	0.9166	16.8011
					MODPSO	26	0.1034	0.2757	1.6535	20.4822
46	80	0.2	4	0.8	MOGA	0	1	3.1673	0.9285	14.8412
					ACO	1	0.9888	4.8402	1.0569	17.3369
					HGA	1	0.9885	3.4941	0.8852	16.3724
					NSGA-II	37	0	0	1.3781	14.455
					MOPSO	1	0.9905	4.1044	0.8703	17.5974
					DPSO	3	0.9677	3.4706	1.0344	15.7246
					MODPSO	39	0.025	0.0534	1.4623	17.359
47	80	0.2	3	0.8	MOGA	0	1	4.505	0.9708	15.1151
					ACO	0	1	5.5505	1.2061	14.971
					HGA	3	0.9559	4.3656	1.5725	17.326
					NSGA-II	23	0.08	0.1529	1.0328	14.6689
					MOPSO	0	1	5.0805	0.8551	17.1628
					DPSO	0	1	5.1013	1.0709	16.5438
					MODPSO	24	0.0769	0.1397	1.5056	17.7407

Problem	*n*	*OS*	*TV*	*FR*	Algorithm	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
48	80	0.2	2	0.2	MOGA	4	0.913	2.0684	1.6592	15.9154
					ACO	3	0.9091	2.4658	1.3728	14.5102
					HGA	1	0.9839	2.4424	0.8596	15.6783
					NSGA-II	30	0	0	1.7193	16.1424
					MOPSO	0	1	2.7966	1.0026	15.6238
					DPSO	2	0.9794	3.1143	0.9997	17.6728
					MODPSO	33	0.0571	0.1475	2.5964	18.3876
49	80	0.2	2	0.3	MOGA	4	0.9437	1.9625	1.1067	17.747
					ACO	1	0.9756	2.2807	1.2162	14.0055
					HGA	3	0.97	2.4942	0.9973	18.2984
					NSGA-II	37	0.0263	0.0152	1.4581	17.0744
					MOPSO	3	0.9714	2.5447	0.9749	17.6966
					DPSO	8	0.9216	2.4286	0.9513	19.31
					MODPSO	43	0.1569	0.2207	1.4082	22.2996
50	80	0.2	2	0.4	MOGA	0	1	2.6435	0.8908	14.3619
					ACO	0	1	3.417	0.9525	13.4585
					HGA	0	1	3.3049	1.1746	15.3427
					NSGA-II	31	0.0313	0.0361	1.7893	15.6519
					MOPSO	0	1	3.4657	0.8214	14.8611
					DPSO	5	0.9438	3.1524	0.9479	16.2461
					MODPSO	32	0.2195	0.3953	1.5176	19.7198
51	80	0.2	2	0.6	MOGA	2	0.9759	2.3906	0.8985	16.3797
					ACO	1	0.975	2.6511	1.4578	14.9682
					HGA	7	0.9167	2.1106	1.1245	17.2041
					NSGA-II	28	0.0345	0.0115	1.2023	16.4548
					MOPSO	5	0.9565	2.4491	0.889	18.0867
					DPSO	2	0.9794	2.3686	0.9751	17.1557
					MODPSO	46	0.0612	0.0282	1.2252	19.652